AD-A268 627

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

ISA-97 COMPLIANT ARCHITECTURE TESTBED (ICAT) PROJECT FINAL REPORT

ASQB-GC-92-006

DTIC
ELECT
S AUG 2 5 1993
B D
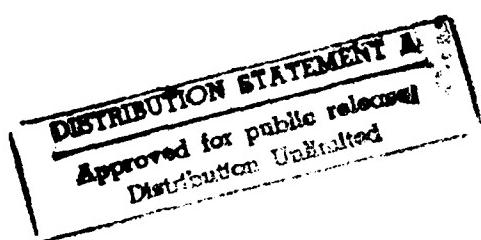
AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800

93-19671

93 8 24 007

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188
Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICAION AUTHORITY N/A	3. DISTRIBUTION/AVAILIBILTY OF REPORT N/A
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ASQB-GC-92-006	5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A

6a. NAME OF PERFORMING ORGANIZATION AIRMICS	6b. OFFICE SYMBOL *(If applicable)* ASGB-GCN	7a. NAME OF MONITORING ORGANIZATION N/A
6c. ADDRESS (City, State, and Zip Code) 115 O'Keefe Building Georgia Institute of Technology Atlanta, GA 30332-0800		7b. ADDRESS *(City, State, and ZIP Code)* N/A
8b. NAME OF FUNDING/SPONSORING ORGANIZATION AIRMICS	8b. OFFICE SYMBOL *(If applicable)* ASGB-GCN	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State, and ZIP Code)* 115 O'Keefe Bldg. Georgia Institute of Technology Atlanta, GA 30332-0800	10. SOURCE OF FUNDING NUMBERS			
	PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO
		DY10	01-01	

11. TITLE *(Include Security Classification)*

ISA 97 Compliant Architecture Testbed (ICAT)   UNCLASSIFIED

12. PERSONAL AUTHOR(S) Melody Eidbo, William Putnam, Michael McCracken, Annie Anton, Paul Cutris, Russell Clark, Mostafa Ammar, George Rouskas

13a. TYPE OF REPORT	13b. TIME COVERED FROM_____ TO_____	14. DATE OF REPORT *(Year, Month, Day)* March 30, 1992	15. PAGE COUNT

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUBGROUP	POSIX, GOSIP, TCP/IP, Distributed Open Environment

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This report describes research efforts into the guidelines provided in the Information System Architecture document prepared by the System Integration Directorate of the USAISEC, August 29, 1992. The report discusses the refinement of the ISA-97 Compliant Architecture Model (ICAM). The ICAM is a modular architecture for the design, implementation of information systems and a guide in the transitioning of Army Information System (ISA) into distributed open environment. The report includes the refinement of the operating system service and interfaces to the other components of the ICAM. The POSIX standard is also examined for suitability as a standard operating systems interface for the ICAT implementation. The report discusses the ICAM demand model that is used to simplify the task of building systems that comply with the requirements of the architecture. The report summarizes the research concerning the transition of the Army Management information systems from a mainframe, flat file, third generation environment to a distributed, open systems environment using workstations, networks, relational database management systems and SQL. The report compares existing military protocol standards to their GOSIP counterparts, in functionality and service, that would be used in a transitioning network environment.

20. DISTRIBUTIONN/AVAILIBILTY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Adrienne J. Raglin	22b. TELEPHONE *(Include Area Code)* (404) 894-3136
	22c. OFFICE SYMBOL ASGB-GCN

DD FORM 1473, 84 MAR   83 APR edition may be used until exhausted.
All other editions are obsolete.

THIS REPORT HAS BEEN REVIEWED AND IS APPROVED

s/ _____

John Gowens, Chief
Communications Network
and Systems Division

s/ _____

John R. Mitchell
Director
AIRMICS

DTIC QUALITY INSPECTED 3

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By_____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

EXECUTIVE SUMMARY

This report describes research efforts into the guidelines provided in the Information System Architecture document prepared by the System Integration Directorate of the USAISEC, August 29, 1992. The report discusses the refinement of the ISA-97 Compliant Architecture Model (ICAM). The ICAM is a modular architecture for the design, implementation of information systems and a guide in the transitioning of Army Information System (ISA) into distributed open environment. The report includes the refinement of the operating systems service and interfaces to the other components of the ICAM. The POSIX standard is also examined for suitability as a standard operating systems interface for the ICAT implementation. The report discusses the ICAM demand model that is used to simplify the task of building systems that comply with the requirements of the architecture. The report summarizes the research concerning the transition of the Army Management information systems from a mainframe, flat file, third generation environment to a distributed, open systems environment using workstations, networks, relational database management system and SQL. The report compares existing military protocol standards to their GOSIP counterparts, in functionality and service, that would be used in a transitioning network environment.

ISA-97 Compliant Architecture Testbed (ICAT) Project

Final Report

Software Research Center

College of Computing

Georgia Institute of Technology

Atlanta, Georgia 30332-0280

CONTENTS

THIS PAGE WAS INTENTIONALY LEFT BLANK

INTRODUCTION

In response to the current information requirements (standards-compliant systems) and strategies (open-systems and decentralization of data processing) of the Army, AIRMICS has proposed the ISA 97 Conceptual Architecture Model (ICAM) as an information system architecture reference model for building an information system using standardized interfaces and components. The ICAM serves as a framework for specifying system requirements and developing portable computer software programs. The establishment of these modules and their application programming interfaces (APIs) will also ease the job of software maintenance, replacement, and information systems acquisition.

As shown in Figure 1, the ICAM consists of six distinct modules: entry, operating system, application, distributed encyclopedia and data, distributed control, and communication modules. These modules are inter-connected by using standardized APIs. This is the model that formed the basis for this project. Each task in the project related to some aspect of the ICAM. Thus the ICAM is the structure which threads throughout the tasks described below.

| Entry		
Operating System	Applications	
	Data Encyclopedia	Data
	Distributed Control	
	Communications Layers	
Hardware		
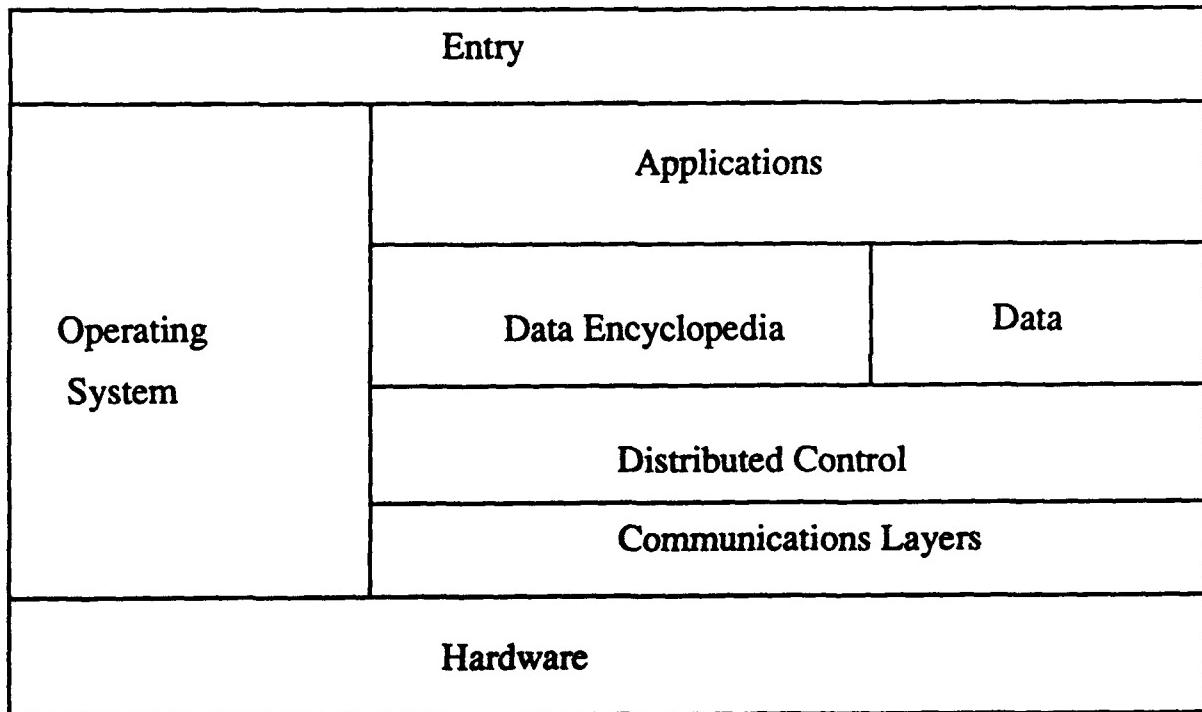
Figure 1. ISA 97 Conceptual Architectural Model (ICAM)

The project was designed to follow the general guidelines provided in the Information System Architecture, Circa 1997 document (prepared by the System Integration Directorate of the USAISEC, August 29, 1990). The primary objective of the project was to test candidate techniques and technologies for incorporation into designs to implement the ISA 97 Architecture.

ISA 97 Compliant Architecture Testbed project was conducted during the period of May through December 1991. The contractor was an interdisciplinary team from the Georgia Institute of Technology, College of Computing and Software Engineering Research Center. The team was divided into small work groups to conduct the research. Each group was responsible for a task and worked on that task independently. This final report is the aggregation of the reports from each of the groups. The project consisted of the following tasks:

Task 1: ICAM Refinement.

This task required the team to (1) define fundamental services offered by each module, (2) provide analysis of the technical accuracy of the ICAM proposal, (3) develop specifications for application programming interfaces (APIs) between the application layer and other layers of the ICAM, and (4) develop protocols for the upper modules of the ICAM (modules above the communication module).

The ICAM is a modular architecture for the design and implementation of information systems and was developed for use as a guide in the transition of Army Information Systems into the distributed open systems environment. The model is conceptual and is intended to represent services required from any information system rather than to provide a design for a particular information system. Using the model as a guide to the required services will also provide a structure from which a design of a system can be initiated. For example, an operating system is a required service; selection of POSIX as the operating system is a design choice with attendant trade-offs. The objective of the ICAM is to assist in evaluating the choice trade-off necessary in designing an information system by defining services parameters required or desired in the system.

Chapter 1 gives the results of Tasks 1 and 2. The first article focuses on the refinement of the operating system services, and interfaces to the other components of the ICAM (Task 1). The article described the separation of application functionality into user interface and computational components and considers the effect of that separation. The article also presents several strategies for further evolution of the ICAM and selection of tool sets commercially available for separating the user interface and the computations components of a system.

Task 2: ICAM Implementation.

Task 2 required the contractor to complete two subtasks: (1) implement the entry layer and its API using X-11 graphical user interface standard and appropriate software to provide lexically, syntactically, and semantically consistent look and feel for all applications such that the end users can access to information services transparently, and (2) provide a consistent operating system interface and common programming interfaces with applications to assure that application programs can be easily ported across the Army's computing environment.

The second article in Chapter 1 gives the results of Task 2. The article focuses on the refinement of the operating systems services. Several application programs are examines to determine services required and operating systems standards are examined to determine how well the operating systems provide the required services. The POSIX standard is examined for suitability as a standard

operating systems interface for the ICAT implementation.

Task 3: Model of Information Services Demand.

The Information Systems Architecture (ISA) defines a modern architecture for U.S. Army Information Systems, consisting of the use of distributed, integrated information systems, that supports an open-systems environment. The ISA is supported by a model that is used to simplify the task of building systems that comply with the requirements of the architecture (ICAM).

Task 3 developed a model for information services demand and recommended metrics to validate it. The model maps user demand for knowledge (as contrasted with data) into the information services defined in the ICAM and addresses demand for information services from the user point of view.

Chapter 2 provides an article on the results of the demand model task. The demand model consists of a generic description language that is then "instantiated" for particular information system configurations, and demands on that information system. It is anticipated that the outputs of this model will be used to refine the ICAM as well as to serve as inputs to performance models. The model is developed as a generic information services demand model and described a method of generating a specific model based on a set of existing and/or proposed information systems.

Task 4: Re-Engineering and Transition.

The three objectives of the Re-Engineering and Transition task were to: identify CASE and re-engineering tools that might be used in transitioning legacy COBOL programs into open system, recommend a least cost strategy for the transitioning databases from flat files into relational databases, and recommend a least cost strategy for the transitioning into GOSIP networks. The results of the task are given in Chapter 3. The first of the three articles in the chapter describes methods for examining applications to determine if the application should be translated, re-engineered, or redesigned into the open environment. A skeleton decision structure is described that allows the criteria to be used. This article also discusses CASE and reverse engineering tools necessary for developing new applications and transitioning current legacy applications to the open systems environment. A CASE tool was acquired for experimentation.

In addition to transitioning legacy software, the task also investigated transition plans for converting to GOSIP communications protocols. The task looked at the existing DOD plan for transition and predicted the best know alternatives for the transition. This issue is still very much open and will be continued in future work. The second and third articles in Chapter 3 discuss the communications strategies.

The second article, A Comparative Study of DOD and GOSIP Protocols, compares existing military standards to their GOSIP counterparts in terms of functionality and services. The network transport, and application layers are considered. Some of the network and transport layer GOSIP standards that do not have counterparts in the existing military suite of protocols are also discussed. The article also addresses some of the interoperability issues and how they will affect the performance of applications.

The third article is a market survey of currently available OSI/GOSIP products that would be useful in transitioning to GOSIP communications. The focus of the survey was limited to the Sun, MS-DOS, and MacIntosh systems. This was considered to be an appropriate scope for the survey, given

3

the potential size of the task if all computer makes and models were included. The article also describes the options for transition from TCP/IP to GOSIP protocols.

It is generally recognized that IP and TCP provide functionality equivalent to that of their GOSIP counterparts. On the other hand, the functions supported by the DOD application layer standards are also supported by the corresponding GOSIP standards. However, the latter offer enhanced services and functions that are beyond the capabilities of the former. The transition to the OSI architecture and the international standards will overcome the limitations of the military standards and will simplify integration of products and expansion to areas such as document architecture and transaction processing that are not addressed or covered by these standards.

In conclusion, the articles published herein establishes that the ICAM model of viewing the information system as a utility which provides generic services to users has support both in the research community and in the commercial community. The ICAT testbed network server well both as an evaluation platform for separation and transition experiments, and as a demonstration platform for open systems applications and products. Implementation of the ICAT network should continue, and its capabilities should be augmented to better represent the open systems environment.

1. ICAM REFINEMENT

THIS PAGE WAS INTENTIONALLY LEFT BLANK

ISA-97 Compliant Architecture Testbed (ICAT) Project ICAM Entry Module Refinement Task

William Putnam

Software Research Center
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Abstract: This document is part of the final report of the ISA-97 Architecture Testbed (ICAT) project. It presents the results of the work on Task 1, which is concerned with the refinement of the ISA-97 Compliant Architecture Model (ICAM), and on Task 2, which is concerned with the implementation of the ICAT itself. The ICAM is a modular architecture for the design and implementation of information systems and was developed for use a a guide in the transition of Army Information Systems into the distributed open systems environment. In this report we focus on the description of the ICAM Entry Module, the services it provides to users and to other modules of the ICAM, and on the implications of the ICAM for the development of user interfaces to information systems. In particular, we describe the separation of applications functionality into user-interface and computation components and consider the effects of that separation. The second part of the report discusses developments in the implementation of the ICAT and presents several strategies for further evaluation of the ICAM and the user-interface/application separation using commercial tools in the ICAT network.

1.0 Introduction

In this task we have worked to refine the ISA97 Compliant Architecture Model (ICAM) developed at AIRMICS. The purpose of the ICAM is to provide an architecture for the development of distributed information systems in an open systems environment. The model is composed of six modules: Entry, Application, Operating System, Distributed Encyclopedia and Data, Distributed Control, and Communications. This report focuses on the refinement of the Entry Module description and on the interfaces between the Entry module and the Applications and Operating System modules.

2.0 Purpose of the Entry Module

The Entry module provides the user with a standardized representation of the information system.

It is the user's interface to the applications, data, operating system, network, and hardware. The primary objective of the Entry module is to provide the user with a transparent and consistent interface to the information system regardless of the application, data, network, or computer being used. Such an interface will reduce training costs, increase ease of use, reduce operator errors, and simplify transition between computer systems or applications.

The issues related to a transparent and consistent interface are:

Transparency

- device independence

- operating system independence

- network independence

- CPU architecture independence

- data format independence

Consistency

- presentation of information

- behavior of the interface

- functional integration of components

2.1 Transparency

Device independence shields the user (and developer) from details of the display and input hardware connected to a computer system. Applications send requests to display information without worrying about byte ordering, pixel sizes, keyboard codes, or cursor controls.

Operating system independence gives users and applications an interface to system resources such as files and applications which hides details such as the storage format of the files, or the binary form of the applications programs.

Network independence gives the users and applications a representation of computing resources in a network which hides location and hardware dependencies. Users do not need to know which devices are attached to which network nodes in order to access and use them. CPU architecture independence allows users to execute applications in a distributed computing environment without concern for processor type. Applications are provided in a binary format which is understandable by all the different computer types in the network, and users don't have to worry about which software runs on which machine.

Data format independence lets applications exchange data across different computer system without concern for conversion issues such as byte order and character sets. Like CPU independence, this is a fundamental requirement for a heterogeneous distributed computing environment.

2.2 Consistency

Presentation of information in an interface may be thought of as the syntax of the interface. In a consistent interface, elements or operations common between components will be called by the same name and displayed or represented in the same manner. In a graphical interface this includes the appearance of objects, for example push buttons and scroll bars. In a command interface it would include names for operations, such as "cut" as the name of the operation to delete or remove an element. The syntax of command options and arguments is also an issue. How are options and

arguments denoted? Is there a required ordering?

In an interface with consistent behavior, a single logical model of operation is used in all applications. If applications are to be activated by clicking on their icons with a mouse, then all applications must have a icon and must respond as expected. If data transfer between applications is accomplished by "drag and drop", then the same method should be used to send files to the printer. Use of devices, such as mice, to select and manipulate objects must also be considered, as does the use and form of menus and selection lists. Specification of program input and output, including redirection and piping is also an issue. Can it be done consistently for filters and interactive applications?

Functional integration means that applications work together as though they are subcomponents of a single, larger system. For example, calendar, scheduling, and electronic mail applications should work together so that a user can easily plan and schedule a meeting. Applications can be integrated by using shared data formats, standard communications facilities, and protocols. Integration allows the use of interface services such as selections and cut/paste operations for information exchange.

2.3 Open Systems Characteristics

An open systems environment is one in which systems conform to published and accepted, or "open" standards rather than proprietary standards. This environment is characterized by its heterogeneous nature. Products from different vendors can work together, or "interoperate", because they adhere to the standards for interaction with other system components. Systems which adhere to open standards for their interfaces with other systems are called "open systems". Advantages of an open systems environment include:

- reduced costs for development and maintenance

- greater portability

- reduced dependence on specific vendors

Three characteristics of open systems (taken from [Nutt90] and [Gray91]) are:

Integration - consistent behavior, presentation, and

functionality

Interoperability - supports exchange of information

among components

Portability - can be moved easily from one environment

to another

In the Entry module we are concerned with each of these characteristics:

  - Integration - we want the user's view of the system to be

    the same across differing hardware and applications

  - Interoperability - we want users to be able to move information

    between objects (including applications) in the interface

    environment

- Portability - we want applications to be developed independent

9

of the details of the interface environment, insulating the

developer from hardware issues

Integration and Interoperability are of primary interest to the users of information systems. Portability is a major issue for systems developers and maintainers.

3.0 Separation of Interface and Computation

It is now generally agreed by user interface researchers that the separation of user interface and computation components of applications is desirable. Advantages of this separation include the ability to modify or tailor an application's interface without affecting the computational part, better support for interface standardization between applications, and greater opportunity for the employment of reusable components. Iterative design techniques can be applied independently to the interface and computation components of an application. [Hart89] [Hurley89]

This separation raises a number of questions, however. Where do we draw the line between computation and user interface? How will the separate components communicate? What impact will separation have on performance? In [Hurley89] the authors point out that a good user interface must have some knowledge of application internals in order to prevent the user from attempting operations which cannot be executed. But this knowledge tightens the binding between the user interface and the application and reduces separation and its associated benefits. So there appears to be a trade off between interface intelligence (an possibly quality) and separation.

The ICAM supports this separation by placing much of the responsibility for user-application dialogue in the Entry module, while the computational components of an application reside in the Application module. Services for information display, input, dialogue control are provided by the Entry module.

But as Hartson describes in [Hart89], the simple view of separation, wherein no computation is done in the interface module, would result in a poor quality interface with little power and intelligence. Input to the application would be in the form of raw information on user actions, such as keystrokes, mouse clicks, and similar events. A great deal of input and display processing would still be required in the application itself.

A better approach is to give the user interface component (the Entry module in our model) enough computational power to handle higher level constructs such as prompts, warning and error messages, help, and some degree of syntactic and semantic checking. Hartson presents a model for this separation which proposes a runtime architecture having four components on three levels. The computation component resides in the application level and represents the heart of the application. At the interface level is the dialogue component of the application, which is subdivided into display and input components. At the hardware level is an interface devices component which deals directly with system hardware to implement display and input operations. The input and display components in the middle layer are endowed with enough computational ability to map raw interface events (such as mouse clicks) into higher level dialogue events (such as button presses).

Our model for the ICAM Entry module is similar to Hartson's but provides additional components in the middle layer for resource access, control, and navigation. We also allow provide support for non-graphical interfaces, though the GUI remains the primary focus of the model.

To relate these models to possible implementations, we can consider the X Window system with the Motif interface environment. X provides mechanisms for the construction and operation of a

10

graphical user interface, but does not dictate any policies on the use of these mechanisms. In other words, it provides tools but no standards for their use in applications interfaces. Motif is an environment which provides higher level interface components for developers and specifies some basic rules for their use in applications. Basic input and display event processing and device interfacing are handled by a separate process called the X server. Mapping of raw interface events into higher level events is handled by the Motif toolkit. The implementation does not separate the interface and computation components of the application as completely as our model suggests, however, because the Motif toolkit functions are linked with the computational component to form the executable application module.

4.0 Interface Domains

In the ICAM model the Entry module has interfaces to two lower level modules: the Applications module and the Operating System module. Following this model, we can divide the problem of a consistent user interface into two domains: Application Interface and Operating System Interface. We will describe these domains, and then consider the interfaces between ICAM modules in this context.

4.1 Application Interface

The purpose of a User Interface is to provide users with a means of interacting with applications. The application does this by displaying text and/or graphics on a computer monitor and by accepting direction and/or information from the user via the keyboard, pointer, or other input devices. When each application has its own unique user interface, inconsistencies occur between applications. These inconsistencies can cause users to make errors, become confused, and will in general reduce efficiency and user satisfaction.

When applications have consistent behavior, users can draw on their knowledge of familiar applications to quickly master new applications or to migrate to new systems. They also make fewer errors and have less confusion when switching between applications in a system. So we see that is important that applications on a given system behave consistently and that behavior is consistent across different computer systems.

To achieve this consistency, application interfaces should be constructed from standard components and follow standard rules of behavior. The standard components are called toolkits, and are typically implemented as function libraries for use by developers. The rules of behavior are sometimes called a style guide, and are provided to developers for reference as they use the toolkit to construct an interface.

The toolkit provides a basic set of interface components such as menus and windows. It also provides mechanisms to manipulate and control the interface components. In a character-based interface the toolkit might include data structures for screens and menus along with functions for cursor positioning, displaying text, and reading text. In a graphical interface the toolkit contains data structures and control functions for windows, scrollbars, buttons, and so on.

Standard toolkits ensure that objects such as menus, windows, icons, buttons, and scrollbars are drawn the same way on all systems. They should also operate in the same way. This is achieved when the toolkit complies with the style guide to implement the specified behavior. But components must be assembled and used in accordance with the style guide in order to be truly consistent. For example, the toolkit provides a mechanism for pop-up menus, but it is the style guide which governs when and how such menus are to be used. There is also the issue of hardware compatibil-

ity. Keyboard mappings, display resolution, color palettes, and other device dependencies must be addressed if an application is going to look and operate consistently on different platforms. Since hardware may vary (the number of buttons on a mouse, for instance) there must be standard translations to a base level environment which can be guaranteed to be available on all systems. All 3 button mice should behave the same way across applications and systems. The translation of unavailable buttons on a 1 or 2 button mouse should be standardized, as well as the use of the keyboard when there is no mouse at all.

In our discussion here we will consider the case of the Graphical User Interface (GUI) and will use the GUI as our model for much of the functionality of the ICAM Entry module. The trend in user interfaces for applications and systems is clearly toward GUIs, and it is also possible to mimic the behavior of a character-based interface in a GUI using a terminal emulator.

4.2 Operating System Interface

The purpose of the Operating System in a computer is resource management. The operating system is responsible the allocation and management of memory, disk storage, processor time, peripheral devices, network ports, and other resources. Commands and utility programs provide the user interface to the management and manipulation of many of these resources. This user interface is typically composed of a command interpreter, or shell, and a collection of utility programs.

The operating system must first be internally consistent. Use of command options and arguments should follow standard rules which apply to all commands and utility programs in the system. But in a heterogeneous network of systems we also need external consistency. The operating systems in the network should all offer a standard set of commands and utilities, at least for common operations.

This does not mean that all computers must run the same operating system. Where the details of the operating system a hidden from the user there will be differences. For example, job scheduling will differ on timesharing and realtime systems, but this is hidden from the user. But the command to rename or copy a file should be the same on either system.

The representation of the file system should be standardized across systems. Executable programs, systems configuration files, libraries, and so on should be located in prescribed places so that users and applications can find them easily.

A degree of consistency can be achieved by buying integrated software packages from a single vendor, whose applications and systems are designed to work together and operate in a consistent fashion. But this approach often locks the user into a relationship with a single hardware or software vendor. It may not be possible to obtain versions of applications software to run on all the types of hardware that may be in use. Also, some functions may not be provided by one vendor's integrated package, requiring the use of different software which does not match the interface behavior of other applications.

A more desirable approach is to standardize as much of the behavior of the user interface as possible, making it independent of any specific application program. This standardized interface should then be implemented across all different hardware platforms which will be employed in the information system. Since it is not possible to divorce all details of user/application interaction from the application program, the application developer should use a standardized toolkit to build applications. The same toolkit is used to implement the application-independent parts of the interface, so that the application fits seamlessly into its environment. Applications from different vendors or

12

developers which are implemented with the standard toolkit will be automatically integrated into a consistent environment.

The POSIX standard which is in development includes specifications for the user interface to the operating system in its Shell and Utilities section. The POSIX Shell and Utilities will serve as the interface between the ICAM Entry module and Operating System module.

5.0 Entry Module Interface Elements

In addition to dividing the domain of the Entry Module into Application and Operating System interfaces, we can divide it into two domains based on the nature of the user/system interaction. When this interaction is conducted though the display and manipulation of graphical objects we are in the domain of Graphical Interfaces. When the dialog is conducted in the form of textual commands given to a command interpreter, or shell, we are in the domain of Command Interfaces. It is possible to have both Graphical and Command interfaces in the same system, and even within the same application, though this presents inconsistencies.

The elements of the two interface domains are:

Graphical Interface

- Device Independence

  - Display

  - Input

- Presentation of Interface Objects

  - Graphics primitives for building objects

  - Standard toolkits for interface development

  - Consistent appearance and behavior of objects

- Direct Manipulation of Interface Objects

  - Window management and operations

  - Icon management and operations

  - Widget operations

- Event Handling

  - Queueing

  - Dispatching

Command Interface

- Command Interpreter

  - standard command set

  - consistent usage of arguments and options

  - interactions between applications

  - error messages and exceptions

- Shell Programming

- standard programming language
- standard environment
- Ownership and Permissions
  - user identification
  - ownership
  - membership
  - access rights (grant, deny, verify)
- Naming, Location, and Navigation
  - file naming requirements and limitations
  - file system representation and organization

Examining these interface domains and their elements, we have identified the services provided by the ICAM Entry module and grouped them into classes.

6.0 Classes of Services - Entry Module

We have defined five top-level classes of services provided by the ICAM Entry module. These classes are: Display, Input, Access, Control, and Navigation. In describing these services, we will use the X Window System as a model. X in an open standard which has become the de-facto standard graphical user interface for UNIX workstations. It is also under consideration for adoption as FIPS 158. We expect that X will be the standard GUI under POSIX as well.

* Display

One of the basic services provided to applications by the Entry module is the display of information to the user. This display may be in the form of Text or Graphics.

- Text
  - Characters
  - Fonts
- Graphics
  - Primitives
    - Lines, Arcs, Shading, etc.
  - Interface Components
    - Windows, Menus, Buttons, Lists, Scrollbars, etc.
  - Resources and Applications
    - icons, composite systems of interface components

The display of information by an application is made possible by the layered set of display services. Using the X window system for an example, the basic display elements of the X intrinsics provide primitives such as lines and rectangles which are used at the toolkit level to construct interface components such as buttons and windows. Toolkit components are then used at the application level to build the application interface, organize the display of information, and structure the interaction between user and application.

In the X model, applications send display requests to a process called the "server", which performs and necessary translations or conversions to handle device dependencies and then displays the information.

Transparency is provided by the separation of the server from the application, or client, code. Consistency is provided by the use of a standard toolkit in the construction of applications.

* Input

Another fundamental service provided by the Entry module is the acceptance of information from the user to be delivered to the application. Input may come from keyboards, pointers, or special purpose devices. The Entry module must handle any device dependencies or translations and deliver to the application the characters, selections, or operations from the user.

- Keyboard Input
  - Text
  - Commands
  - Functions
- Pointer Devices
  - Direct Manipulation
  - Selections
  - Focus

Some processing of the input stream may be done to ensure correctness (parity checking, byte ordering) or compatibility (newline-carriage return conversions, control character mapping). In a multitasking environment where several applications may be active each application will have its own input stream or queue which must be managed.

In the X Window System model, basic processing of the input stream is handled by the X server process. The server again handles all device dependencies, and dispatches input to applications in the form of "events". Each application has an "event queue" which contains its input stream.

* Access

One service of the Entry module which can easily be taken for granted is the provision of access to objects in the computing environment. Objects must be named and identified, and typically have attributes such as ownership and membership which must be observed. This service is a composite of the lower level services provided by the Operating System for resource management:

- Identification
- Ownership
- Membership
- Verification

* Control

The user must be able to manipulate objects in the computing environment. Once the objects are identified, the user will need to be able to start and stop applications, use an application on a particular data object, and, in a multi-tasking environment, switch between applications.

Some functions provided by the operating system command interface may be provided graphically, including copying, movement, and deletion of files or other interface objects.

- Manipulation of Interface Objects

  - Activat. n

  - Selection

  - Cut & Paste

  - Drag & Drop

- Manipulation of Applications

  - Start, Stop, and Suspend

  - Connection (pipes and redirection)

  - Foreground / Background

  - Shell Programming

* Navigation

The user must be able to navigate in the computing environment to locate applications, data, and resources to be manipulated. This may be done using operating system commands to traverse the file system or it may be done graphically using a browser and iconic representations of system objects and resources. When the user is interacting with an application which has multiple components, it is necessary to move the focus of attention among the components to carry on a dialog with the application. This may be viewed as a form of navigation within the components of an application and is similar to the process of navigation between applications and other elements of the computing environment.

- Navigation in the Network

  - host identification

  - network management

- Navigation in the File System

  - representation of the file system

  - organization of objects within the system

- Inter-Application Navigation

  - focus on active application

- Intra-Application Navigation

  - focus on active component

7.0 ICAM Testbed Implementation

As part of the validation of the ICAM, we are setting up a testbed network at AIRMICS using open systems standards and applications to connect a heterogeneous mix of computer systems. When complete, the TCAM testbed network (called ICAT) will be used to demonstrate open systems principles and applications in the Army environment.

Most of the effort during this project went into two areas: product research and evaluation and soft-

ware installation and configuration. In the area of product research and evaluation, we gathered information on open systems products in the areas of operating systems, user interface software, and networking systems. This report will describe the user interface development and transition tools and packages which were identified or installed. Software for operating systems and networking are described companion reports.

7.1 ICAT Hardware Configuration

The ICAT network is currently part of the AIRMICS ethernet LAN. It was intended that the network would be separated from the administrative part of the AIRMICS LAN and connected with that network and the internet using a gateway system. The hardware for the gateway (a second ethernet card in a SPARC workstation) is installed, but the computers designated for the ICAT subnet have not yet been reconfigured for use with the thin ethernet cable to be used for the subnet. Reconfiguration requires the removal of the PC ethernet cards for the installation of hardware jumpers. Cabling and configuration for the ICAT subnet should require about 1 week to complete.

7.2 Software Installation and Configuration

Network software configuration is partially complete. The configuration for the network gateway (the router) must be determined, and the addresses of the ICAT subnet machines assigned when they are re-wired for the subnet. PC-NFS software is installed and operational on all the PC systems to be used on the ICAT subnet.

One of the primary objectives of the ICAT network is to demonstrate a consistent user interface on heterogeneous systems. This is being approached in two ways. For graphical user interface (GUI) capability we are installing the X Window System with the Motif toolkit on both UNIX and DOS systems. The base X distribution was installed and made operational on all SUN systems at AIRMICS, including SPARC, Intel 386i, and Motorola 68020 architectures. An X product for MS-DOS PCs, called PC-Xview, was purchased and installed on AIRMICS PC systems, but could not be made operational before the end of the project due to network configuration problems. These problems have been resolved, and it should be possible to bring up the system with a few hours of work.

The Motif toolkit distribution from the Open Software Foundation was obtained through Georgia Tech and installed on AIRMICS Sun SPARC systems. Motif is a "look & feel" standard for GUIs which goes on top of the X Window System. We intend to use Motif in the development of a graphical interface for a STAMMIS - most likely the IMCSRS system which the COBOL re-engineering and transition group has been examining, but possibly other systems as well. We also ordered a Motif-based GUI development tool called Builder Xcessory which will be used with the Motif libraries to develop application interfaces.

To demonstrate a consistent command interface on both UNIX and DOS systems a product called PolyShell was purchased and installed on AIRMICS PCs. This product is a UNIX work-alike system which runs on top of DOS and provides the UNIX C-shell command interpreter and many of the standard UNIX commands and utilities. In addition, the PC-NFS package which was installed to network the DOS machines with the UNIX workstations offers standard UNIX commands for file transfer, terminal emulation, and remote job execution.

While examining the separation of interface and computation we have identified some commercially available products which claim to provide this capability in different ways. A system called XVT from XVT Software provides an API for graphical interface development which can be

linked with different runtime libraries to form executable versions of an application using different interface standards, including X, Macintosh, and Microsoft Windows. A product called Deskterm from IXI Corporation provides two approaches: an API which transforms character based operations into graphical operations and a script based front end for existing applications which places a graphical interface on top of a character based application. Both XVT and the Deskterm API require source code modification and linking with interface specific runtime libraries. The Deskterm Soft Option (the script interpreter) does not require access to or modification of application source code. A third product, Oracle Tools, offers an environment for the development of graphical interfaces to applications based on the Oracle relational database system.

These products offer the opportunity to examine three different approaches to user interface transition. The first approach is to take an application and modify the source code to make it use an intermediate API such as that provided by XVT or Deskterm Protocol. The second approach is to add a graphical interface without modification using a script interpreter such as Deskterm Soft Option. The third is to first transform the application into an RDBMS application and then add a graphical interface using a development system such as Oracle Tools.

These approaches could be compared with the process of building a graphical interface for an application directly, using GUI development tool. Tools such as Builder Xcessory offer rapid prototyping and development environments for applications using the X Window System.

8.0 Conclusions and Recommendations

We have established that the ICAM model of separation of user interface and computation components of applications has support both in the research community and in the commercial community. The exact nature and impact of that separation is still and open issue, however. The commercial products described above present the opportunity to evaluate and compare several approaches to the separation. We feel that this evaluation would be very helpful in the development of strategies for the transition of Army information systems to the open systems environment.

The ICAT testbed network will server both as an evaluation platform for separation and transition experiments, and as a demonstration platform for open systems applications and products. Implementation of the ICAT network should continue, and its capabilities should be augmented to better represent the open systems environment.

18

References

[Foley90]James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. Computer Graphics, Principles and Practice, Second Edition. Addison-Wesley,1990.

[Gray91]Pamela Gray. Open Systems, A Business Strategy For The 1990s. McGraw-Hill, 1991.

[Hart89]Rex Hartson. User-Interface Management and Control. IEEE Software, January 1989, pp. 62-70.

[Hurley89]William D. Hurley and John L. Silbert. Modeling User Interface-Application Interactions. IEEE Software, January 1989, pp. 71-77.

[Jones89]Oliver Jones. Introduction to the X Window System. Prentice-Hall, 1989.

[Mayr90]Anneliese von Mayrhauser. Software Engineering, Methods and Management. Academic Press, 1990.

[Nutt90]Gary J. Nutt. Open Systems. Prentice Hall, 1990.

[MPG91]Open Software Foundation. OSF/Motif Programmer's Guide, Revision 1.1. Prentice-Hall, 1991.

[MSG91]Open Software Foundation. OSF/Motif Style Guide, Revision 1.1. Prentice-Hall, 1991.

[MUG90]Open Software Foundation. OSF/Motif User's Guide, Revision 1.0. Prentice-Hall, 1990.

[Powell90]James E. Powell. Designing User Interfaces. Microtrend Books, 1990.

[Young90]Douglas A. Young. The X Window System: Programming and Applications with Xt, Motif Edition. Prentice-Hall, 1990.

Commercial Products

The following products have been identified and/or acquired for evaluation in the ICAT testbed:

PolyShell from Polytron Corporation, Beaverton OR, 503-645-1150 UNIX work-alike for DOS systems with C shell, possibly POSIX compatibility for DOS - installed

Poste from Alfalfa Software, Cambridge MA, 617-497-2922Electronic mail system with X-Motif compliance and X.400 support

PC-Xview from UniPress Software, Edison NJ, 800-222-0550X Window Software for DOS Personal Computers

Builder Xcessory from ICS Incorporated, Cambridge MA, 617-621-0060GUI development environment for X-Motif applications

XVT from XVT Software, Boulder CO 303-443-4223 GUI development kit - API that's look & feel independent, let's you develop to API, then link to chosen GUI standard

DeskTerm from IXI Corporation, San Ramon CA, 510-275-3120 Development kit for adding Motif front end to character based applications, including RDBMS systems, with or without source code modification

Oracle Tools from Oracle Corporation, 800-633-0521 GUI development kit that goes with Oracle RDBMS, maybe a good choice for COBOL transition

ISA-97 Compliant Architecture Testbed (ICAT) Project ICAM Operating Systems Services Refinement Task

Melody Moore Eidbo

Software Research Center
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Abstract: This document is part of the final report of the ISA-97 Architecture Testbed (ICAT) project. It presents the results of the operating systems work for task 1, whose overall goal is refinement of the ISA-97 Compliant Architecture Model (ICAM), and task 2, and on task 2, which concerns the implementation of the ICAT testbed itself. The ICAM is a modular architecture for the design and implementation of information systems and was developed for use as a guide in the implementation of the Army's systems to a distributed open systems environment. This report focuses on the refinement of the operating system services, and interfaces to the other components of the ICAM. The POSIX standard is examined for suitability as a standard operating systems interface for the ICAT implementation.

1.0 Introduction

With the advent of new technology in computer communications, it is not uncommon for large information systems to be shared by users on multiple machines. The trend towards the desktop workstation encourages a distributed computing model rather than the single, large mainframe with terminals, as has been used in the past. Distribution has many advantages, among them accessibility, reliability, and increased computing power.

However, with this distribution comes heterogeneity in hardware platforms. The availability of many different architectures means that it is likely that a computer system will consist of several different platforms, rather than one single one. This produces problems with interoperability and porting software from one machine to the next; differences in local operating systems and command interpreters means that users must learn a new interface for each machine.

The recent philosophy of Open Systems seeks to solve these interoperability problems. By agreeing upon a standard interface, and implementing that interface on multiple platforms, applications can be easily ported from one machine to another. The Open Systems effort, supported by such

organizations as IEEE, ISO, and ANSI, has been defining standard interfaces for many facets of development, including operating systems, user interfaces, communications, and language interfaces.

The work in this task of the AIRMICS ICAT project has been to identify operating system requirements for the STAMIS information system applications and to refine a model of services that should be supported. These goals were augmented by a desire to provide the maximum portability among several hardware platforms. Therefore, a natural source of information and input has been the Open Systems Standards efforts. We have studied in particular the POSIX Operating System interface as the base of our system services definition.

2.0 The OS Model within the ICAM

The ICAM model defines a series of components that interact with each other to form a system. (See Fig. 1) Each juncture between components defines an interface. The scope of the work presented here involves the interface between the Operating Systems component and the Entry Layer component (called the Command Interpreter), and the interface between the Operating Systems component and the applications (called System Services).

In the model of the operating system (see Fig 2), the process primitives provide a base for other system services. These primitives form the low-level kernel services for process control, synchronization, and interprocess communication. Sharing the base of the model with the process primitives is the file system. File System primitives include file control, organization, and navigation.
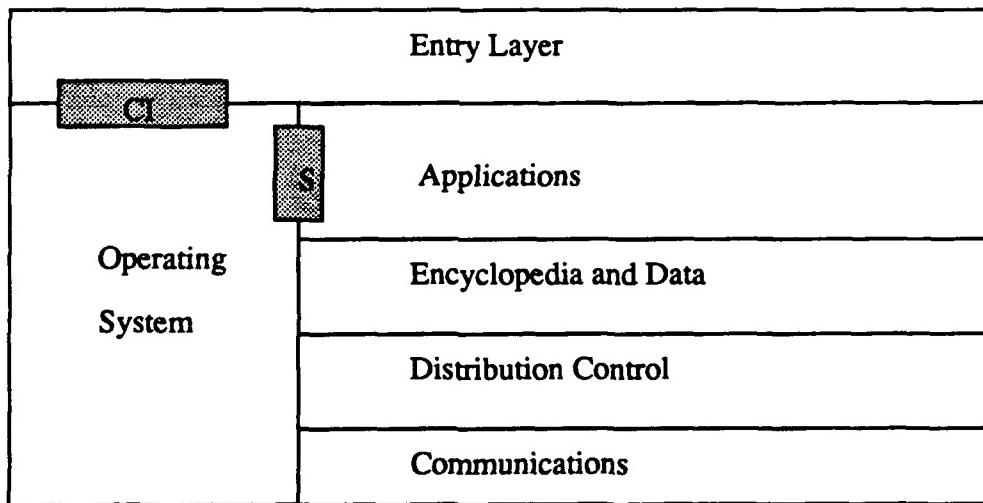
Figure 1: The ICAM Model with OS Interfaces

Upon the base of process and file primitives, the rest of the operating systems model is built. I/O services, such as read, write, and pipes, use the process primitives and file primitives to implement higher-level functionality. Similarly, the process environment, including job control, process identification, and rudimentary security, are built from the same underlying primitive units. Device drivers provide interfaces to external device hardware using the building blocks of the file and process primitive

22

bases.

The Command Interpreter incorporates all of the other operating systems components, giving the user access to all of the supported services. The language support libraries perform the same service for applications programs, providing an interface to operating system functions from application code.

Command Interpreter				
File System	I/O	Process Environ - ment	Device Support	
Process Primitives				
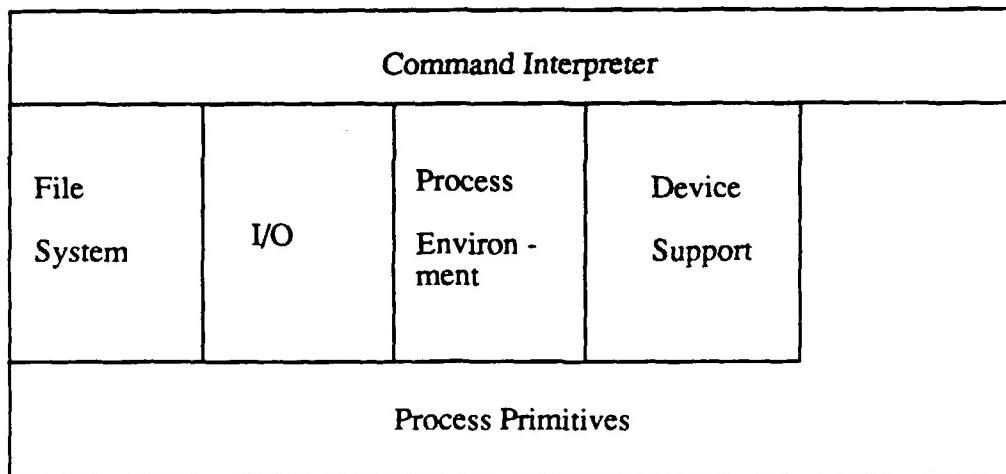
Figure 2: The Operating Systems Services Model

3.0 Command Interpreter Requirements

The POSIX services model is based upon Unix, therefore, as in Unix, the command interpreter function of interacting with the user is performed by a program called the shell. There are also utilities associated with the shell to form a toolset for the user. Currently, a standard is being developed for the POSIX shell by IEEE Working Group 1003.2. This standard is nearing completion but is still under review.

The 1003.2 standard defines a programming, or source code level interface to operating system services. The shell offers interactive communication between the user and the system, and also programmability for longer or repeatable tasks (shell scripts). In the ICAM model, the command interpreter is part of both the operating system component and the entry layer component.

Although the command interpreter standard is not completed, there is a set of requirements that has been refined. Following are the command interpreter requirements for the ICAT:

* A shell command language and interpreter that allow users to write shell scripts.

* An interactive command language that allows the user to execute programs, perform I/O redirection, handle command line arguments, and other shell control functions.

* A paradigm for argument specification and command names.

* Primitives to allow applications programs to access shell facilities, such as command parsing.

* Environment variables that can be created and manipulated directly by the user, or from applications programs.

* Access to file system directory hierarchy and organization utilities.

23

* Complex or high-level data manipulation utilities that can be accessed from applications programs.

* Language services support, currently the languages C, Fortran, and Ada are being considered.

* Standard installation procedures for applications.

* Privileged operating system commands for administration, such as creating user IDs, initiating daemons, etc.

4.0 Operating System Services Definitions

POSIX Standard 1003.1 was published in 1988; it describes a fairly complete operating system interface that is an approved standard. For this task, we identified the Operating systems service requirements for the STAMIS applications and mapped them into this standard.

The Operating System services are divided into several areas of

functionality and level:

* Process Primitives

* Process Environment

* File System Primitives

* Input / Output

* Device interfaces

* Language Support

The Process Primitives and the File System Primitives are used by the other components of the operating system as building blocks. Process and file primitives may also be directly accessed by the user or application programs. Following are the Operating System requirements for the ICAT, and a mapping to the POSIX standard 1003.1 where supported.

4.1 Process Primitives

Process primitives are typically kernel functions that involve process control, interrupts (or signals), timing, and communication. The services required by the ICAM are described below.

- Process Creation - In Unix, this process (fork) duplicates the parent process to produce a child process, thereby "creating" a new process. The child process inherits the parent's process environment and file descriptors, but these can be modified.

- Execution - the Unix exec function is supported. This allows a new process to be created, and then substitute a new process environment to result in a completely new process. The POSIX exec function supports arguments to the new program.

- Termination / Deletion - Processes can terminate normally (by a return from the main routine), or abnormally (from an error or an untaught signal). Processes can also be aborted. It is also possible to wait for another process to terminate.

- Synchronization - Unix uses the signal as a synchronization primitive, and POSIX also supports them. Signals are software interrupts that can be generated by processes and may usually be trapped by other processes. There are some signals (such as kill) that cannot be trapped, ensuring that processes can be destroyed if they get hung or deadlocked.POSIX supports an extensive range of signal functions, but it does not provide any higher-level synchronization such as semaphores.

24

- Communication - POSIX does not directly provide a model for communication between processes. Unix process communication consists mainly of signals and shared memory capabilities. There is are no message primitives. The ICAM communication requirements could be met by a combination of signals and pipes (see the I/O section). This would be a good area for further study and exploration.

- Timing operations - POSIX provides an alarm function that allows a process to "set" an alarm and then receive a signal after the designated time has elapsed (an interval timer). The alarm function is not completely accurate, since it does not compensate for process scheduling time. However, it should be sufficient for ICAT requirements.

4.2 Process environment

The process environment involves process identification (for security and file access), user identification, and job control. Following are the process environment requirements for the ICAM:

- Process ID - Process ID's are assigned by the system in ascending chronological order. POSIX provides several functions to obtain the process ID of a process or its parent.

- User ID - User identification is important to process and file security. Users may be granted access to a file through permissions on the real user, effective user, real group, and effective group ID's. POSIX provides functions for these as well as a setuid function. User names and group IDs can also be obtained.

- Job control - POSIX provides functions for job control, including setting process group IDs to determine security access to files and other processes.

- System identification, system name - POSIX supports functions that allow an application program to determine information about the hardware platform it is running on. These functions can greatly enhance portability.

- System time - System time is a monotonically increasing counter that allows relative time values to be computed. POSIX supports the time function to retrieve this information, and also to record information about process timing.

- Environment variables - Environment variables allow processes to set, test and modify globally accessible values. POSIX supports the Unix getenv function, which searches the environment variables for the specified name, and returns a pointer to its value. Environment variables can be used as a rudimentary form of communication.

- Terminal identification - Knowing the characteristics of the user's terminal can be useful for an application user interface. POSIX provides functions to retrieve information about the terminal, including the device name and terminal pathname.

4.3 File System

File primitives are important to many components of the operating system. They are used to store information about processes, to implement input/output, and to effect communication as well as store information directly from the user. As in Unix, the POSIX file organization is hierarchical, with files as the leaves of the tree and directories as the nodes. Following are the file systems services required by the ICAM:

- File manipulation - POSIX supports a general set of file manipulation functions, including open, close, create, delete, append to an existing file, reset a file, associate an external filename with an

internal filename, and creating special files. POSIX also defines a standard file characteristic profile, including header and data structure standards.

- Directory operations - POSIX supports a collection of directory operations, including opening, adding entries, closing, reading, and deleting.

- File security - POSIX provides functions to set the owner and group IDs for files to restrict access. File access and modification times can be recorded.

- Navigation within filesystem - POSIX provides a set of functions that allow the user to find files within the filesystem. Among these functions are change working directory (which changes the current default directory), and obtaining the current working directory pathname.

- File characteristics - As in Unix, POSIX provides functions to set certain attributes of files, such as file permissions, input/output modes, and file ownership.

4.4 Input/Output

The file system provides the support and storage for input/output activities. The I/O functions are built on top of the file primitives to provide higher-level functionality. Following are the ICAM requirements for I/O:

- Pipes - A pipe is an inter-process communications channel. A single pipe has a read end and a write end; therefore, information flow is one-way along a pipe. Information can be buffered inside of a pipe. Two-way communication can be achieved by using two pipes. POSIX supports the Unix pipe function.

- File descriptors - POSIX supports file descriptor manipulation functions, such as duplicate and deassignment (close a file).

- Read from file - POSIX supports the unix Read function, which reads a specified number of bytes from a specified file.

- Write to file - POSIX supports the Unix write function, which attempts to write a byte buffer to an open file descriptor.

- File control - Files can be locked for reading or writing, preventing accidental access or destruction of files.

- Reposition / Reset / Set offset - POSIX allows further control, especially with special files, to change the position of the read or write within the file.

4.5 Device Functions

In a heterogeneous environment, it is difficult to standardize on device interfaces. Devices tend to have many special purpose constraints and must be dealt with at a rather low level. However, POSIX addresses these issues and provides some standards for asynchronous communication ports and terminal control. Following are the device-level requirements of the ICAM:

- General Terminal Interfaces - POSIX has a set of functions that support terminal device files. Among the functionality provided:

  * Open a terminal device file - Terminals are usually opened by special processes, and applications processes inherit a "standard" input, output, and error file. However, POSIX does give the control to explicitly open a terminal file.

  * Close terminal device file - When the last process that could send output to the screen termi-

nates, the terminal device performs a disconnect.

* Process groups - terminals may have processes associated with them, which play a role in handling interrupts. Process groups assigned to terminals may be manipulated, such as killing them all if the terminal file is closed.

* Access control - Since processes may run logically in parallel, there must be some mutual exclusion mechanisms for writing to the terminal (otherwise output from different processes could intersperse on the screen). POSIX provides access control that blocks background processes from writing to the terminal device.

* Data control - Since terminals can operate in full-duplex mode, it is possible that input can be accepted at the same time output occurs. This requires input queueing. POSIX provides two modes of input process buffering, canonical and non-canonical. POSIX also provides for echoing input to the screen for full-duplex mode.

* Write control - Similar to input buffering, some implementations may buffer output to the screen. POSIX provides a standard mechanism for write buffering.

* Special characters - Some characters have special effects on a terminal screen. POSIX provides a set of standard special characters that can be generated and received by processes.

* Control modes - POSIX provides a mechanism for setting fields in a flag to control the operation of the terminal hardware (such as a setup). These terminal characteristics can include receiver enable, number of bits per byte for a character, stop bit parameters, and parity information.

* Baud rate - POSIX supports changing and obtaining information about terminal baud rate.

* Terminal interface control - POSIX provides a large set of terminal control functions. Among them are: set and get state information, line control functions (such as send break), process group ID functions, and foreground and background control.

4.6 Language support

The POSIX standard 1003.1 (1988) specifies language support interfaces for the C language. Nearing completion [IEEE91] are the specifications for an Ada language binding and a Fortran interface to POSIX system services. The services described in this section are general to all languages; we will be most interested in the Ada language bindings. Following are the language services that are required by the ICAM:

- Math functions - a full math library (including trigonometric functions, log and natural log, exponentiation, square root, etc.) should be supported.

- Input/Output - System services for file access, pipes, read, write, file control, naming, and file characteristics control are required.

- Date and Time - The system clock should be accessible for timing functions.

- Access to system primitives - There should be an interface through which application programs can call system primitives such as process control and file control.

References

[Kuhn91] Kuhn, Richard D. "IEEE's POSIX: Making Progress", _IEEE Spectrum_, December 1991, p 36 - 39.

[IEEE88] IEEE Standard Portable Operating System Interface for Computer Environments (POSIX 1003.1), Institute of Electrical and Electronics Engineers, Inc. 1988.

[GCN91] Miles, J.B., "POSIX Software", Government Computer News, Sept. 16,1991.

ICAM REFINEMENT

THIS PAGE WAS INTENTIONALY LEFT BLANK

The ICAT Information Demand Model

W. Michael McCracken
Annie I. Anton
S. Paul Curtis

Software Research Center
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Abstract: The Information Systems Architecture (ISA) defines a modern architecture for U.S. Army Information Systems, consisting of the use of distributed, integrated information systems, that supports an open-systems environment. The ISA is supported by a model that is used to simplify the task of building systems that comply with the requirements of the architecture (ICAM). The ICAM demand model task has developed a service based model of interaction with proposed and existing systems. The model consists of a generic description language that is then "instantiated" for particular information system configurations, and demands on that information system. It is anticipated that the outputs of this model will be used to refine the ISAM as well as to serve as inputs to performance models.

1.0 Introduction

The Information Systems Architecture, Circa 1997 (ISA97)document defines the modern architecture for U. S. Army information systems. Some objectives are (1) the use of distributed, integrated information systems (2) within an open-systems environment.

With this in mind, AIRMICS has developed an architectural model of informations systems, called the ISA97 Compliant Architecture Model (ICAM), that will simplify the task of building ISA-compliant information systems.This model identifies the major components of information systems and describes the interaction between these components.Furthermore, the ISA97-Compliant Architecture Testbed (ICAT) is being develope to refine and partially implement the model.

The purpose of Task 4 is to develop a modeling technique to model the demands on information services in U. S. Army information systems. We have developed a generic information services demand model and described a method of generating a specific model based on a set of existing and/or proposed information systems.

31

1.1 ICAT and the Demand Model

The ICAM may be very useful in developing ISA97-Compliant information systems, but first it must be refined and validated; the ICAT will help to do this. To build the ICAT testbed, much of the ICAM model must be refined. In addition, many of the technical feasibility issues of the ICAM will be answered by the testbed. The ICAT will not, however, indicate how well the ICAM model will scale up to larger information systems. The ultimate goal of the Army using ICAM is to integrate many large U. S. Army information systems into a single distributed system.

To help assess the applicability of the ICAM for systems larger than the ICAT, a technique to model information services demand has been developed. Once a demand model has been created, the ICAM can be evaluated for scalability to that particular size. That is, the ICAM can be assessed in reference to a particular set of demands for information services. If a performance model exists for the ICAM, the information service demand model may to used to drive it.

This demand model is similar to a system requirements model. A requirements model is an external--or user oriented--view of the system. Certain characteristics of the system's required functionality are modeled, often including characteristics that affect how the system must be implemented. A demand model is also an external view of the system. It too models certain characteristics of the system's required functionality. The demand model, however, is concerned with characteristics that affect the scalability of the ICAM--not its implementation.The proposed method of constructing demand models borrows from the requirements modeling literature [Ros85]. Some characteristics needed for requirements modeling, however, are not needed for the demand modeling. Also some characteristics needed for demand modeling are not traditionally included in requirements models.

1.2 Demand Modeling Technique: Representation and Method

Our research has been focused into two areas---defining the generic information services demand model and developing a method to instantiate a model. In defining the generic model, we have specified which characteristics to model and how to represent that information. The modeling method describes how to gather and transform the necessary information into the final representation.

In Section 2.0, we describe the model and its representation.That is, we describe a template which can be used to instantiate demand models.

In Section 3.0, we describe how to create a demand model for a particular system or set of systems. That is, how the data is gathered and used to instantiate a demand model.

2.0 Demand Model

The ICAM describes how applications will reside within and interact with the other information system components. The information services demand model captures some characteristics of the demands made by an ICAM application on the rest of the ICAM architecture.

A demand model may have several uses. It may be used to help design and tune an information system. It may also be useful to generate test data for testing of the information system; particularly for statistical usage testing as described by Cobb and Mills [CM90]. But the primary use of a model would be to assess the scalability of a proposed system architecture against a specific set of information services demands. This, would be particularly useful if a performance model of the proposed architecture could be driven by this model. The scalability assessment has been the driving

force behind the approach and details of the demand modeling technique.

2.1 Model Approach

The approach to developing the demand model was divided into three subtasks. The first subtask task was to determine which characteristics of the system to model. There are many characteristics about the interface between an application and the rest of the information system architecture, but which of these effect the scalability of the system? The second subtask was to determine how and at what level to represent these characteristics. The final subtask was to develop an overall structure within which to represent these characteristics.

2.1.1 Characteristics Modeled

We know we want to model demand for information services. But what is relevant about these demands do we model? There are certainly many things that characterize information services demands, but which of these might affect the scalability?

According to ICAM, an application, and indirectly a user, interfaces with the rest of the system through three application program interfaces (APIs):

- the entry layer,

- the operating system layer, and

- the distributed encyclopedia.

All three of these components provide information services to an application program, and the functionality of all three interfaces needs to be examined to help validate the ICAM. But only the distributed encyclopedia is important in validating the scalability of the ICAM.

The entry layer for each site should be independent. As the number of sites in the information system grows, the entry layer will remain unaffected. The entry layer will be validated by the ICAT, and it will not affect the scalability of the system. Therefore, the interaction between an application and the entry layer is not important for the demand model.

The operating system layer for each site is also relatively independent. The operating system may provide applications some limited communication to other sites (via electronic mail or file transfer protocol), but, for the most part, the operating system layer at a particular site will not be affected by other sites. Therefore, the demand model will also ignore interaction between an application and the operating system layer.

The ICAM distributed encyclopedia component at a site is highly sensitive to other sites. In fact, in ICAM there is just one distributed encyclopedia that is spread across all the sites. As an information system is scaled up, the requirements for the distributed encyclopedia change. The ICAT may validate some of the functionality of the distributed encyclopedia component, but it cannot validate its scalability. The demand model will be exclusively concerned with the interaction between applications and the distributed encyclopedia.

The distributed encyclopedia provides transparent data access. Access to data is then the primary characteristic to be modeled. Because different concurrency control mechanisms are used, it is important to distinguish between updates and inquiries. It is important to model how much data is being accessed and how often. Finally, it is important to model where the data is being accessed from.

33

2.1.2 Level of Model

The model must capture most relevant information about the system's scalability without being concerned about low level details.

The model treats the distributed encyclopedia as a black box. At this point in the development of the ICAM, the details of the distributed database implementations are unknown. We cannot therefore, model all the processing and communications going on inside the distributed encyclopedia. The logical structure of the database has not been specified, so we cannot model the logical database processing (such as indexing, joins, selections, etc.).

The model will represent high-level accesses to data objects, without reference to the logical or physical structure of the distributed database.

2.1.3 Transaction Basis of Model

Information systems are essentially event-driven; the events are translated into transaction requests. The transactions are passed into demands onto the ICAM distributed encyclopedia component. The information services demands can then be expressed in terms of required transactions.

2.2 Model Overview

The demand model characterizes information services demands as a set of transactions on data items. Each transaction consists of the following elements:

<transaction_name>: A unique identifier to distinguish transactions.

<frequency>: Numbers indicating the average (and optionally peak) frequency with which a transaction will be initiated.

<location>: A unique identifier for the site which requests the transaction.

<reads>: A list of data items that this transaction reads from the distributed encyclopedia. Associated with each data item in this list is an estimated number of tuples read.

<writes>: A list of data items that this transaction modifies in the distributed encyclopedia. Associated with each data item in this list is an estimated number of tuples written.

Note that by analyzing the transactions and data items, the resulting demand model may be refined. As transactions are optimized, common data items integrated, etc, the demand model will more accurately model the specific information services demands. We also note that the term tuple has a formal definition of a group of related fields in a row of a relation. We are using the term more informally, to note a group of related fields. This is used to help avoid the issue of defining data. For example, we can use an employee tuple without being specific as to its content, but it can be easily understood by both the user and the modeler.

2.2.1 Data Item Reads and Writes

The "data items" are high-level objects stored in the data base. The demand model is not concerned with the logical structure of the relations within the information system. Instead, it is concerned with data objects from the application program's view.

Data items are defined to completely reside in a single location. It is also required that the data items have consistent names across transactions and applications.

The "reads" indicate data read from the distributed encyclopedia, but not modified. The "writes" indicate that data is modified in the distributed encyclopedia. In some cases, data will be read and

then written.

The model is most useful if there is some indication of the amount of data associated with each read and write. It is not necessary to have an exact number, but an average figure (or estimate of an average figure) is useful. The demand to return a single tuple is quite different that the demand to return 10,000 tuples.

2.2.2 Transaction Location

Although the ICAM specifies a distributed encyclopedia which removes concerns about data locality from applications, the actual implementation of a distributed information system must be concerned about data locality. Therefore, it is important that the transaction origin be modeled.

The location should correspond to the physical sites of the distributed information system, not just to U. S. Army installations or functional organizations. For example, there may be several information system sites at any particular fort. It is important that a consistent naming scheme be used to identify locations.

The details of this naming scheme have not been identified yet. It is assumed, however, that the U. S. Army already has some naming scheme to identify organizations and their locations that can be adapted to specify transaction locations.

It is possible to list a transaction location as "unspecified." There could be two possible reasons for this. Firstly, the originating location of the transaction may not be known by the persons building the demand model. Secondly, if a transaction may occur from a number of different sites, the location may be left "unspecified" to keep from having to list a separate transaction from each site.

2.2.3 Transaction Frequency

The type of transactions that may occur between an application program and the distributed encyclopedia does not fully specify the demands on the distributed encyclopedia. The frequency of transactions is also necessary. Each transaction in the demand model, then, has an associated frequency associated with it that indicates when the transaction occur.

The average frequency of the transaction is adequate for all but very detailed assessments. The average frequency is represented as the number of transactions initiated per day. If the originating location of the transaction is "unspecified" because it may be originating from multiple sites, this indicates the average number of transactions initiated per day from all the sites.

For more accurate representations of the information services demands, the peak load may also be represented. A particular transaction may only occur an average of ten times a day, but it may actually occur three hundred times a day on the last day of each month. To represent the peak load, the peak transaction frequency and the peak time and duration must be specified.

If this model is used to drive a performance evaluation tool, the peak time and peak duration would need to be more formally defined. In fact, the transaction frequency could be further refined in a number of ways.

For instance, the frequency could be specified as a continuous function indicating the probability of a transaction being initiated at any particular time. For more simple assessments, however, just the average transaction frequency will suffice.

3.0 Demand Modeling Method

The information services demand model should map user demand for information into the infor-

mation services defined in the ICAM and address demand for information services from the user point of view.

In order to understand the demand services needed by users, the requirements must be acquired in some way. This section describes the process for instantiating a demand model. The demand modeling instantiation method can be decomposed into phases. The first phase entails actually gathering the requirements and the second phase addresses the actual requirements modeling process.

In terms of gathering requirements (for transition to the model), we are concerned with gathering the following information:

<reads> (and amount of data)

<writes> (and amount of data)

<transaction_name> (i.e. process)

<frequency> (of transactions average, or peak)

<location> (locality of transactions request)::(special = unspecified)

Several possible approaches exist for gathering this information, including: a study of current information systems, conducting interviews throughout the user community, or by gathering the requirements using a group requirements determination technique. We suggest group requirements determination as the most appropriate technique based on the literature which presents the ARMYs previous successful experiences in using this requirements elicitation technique [DDH91].

3.1 Requirements Elicitation

Traditional requirements elicitation techniques have included: sequential interviews of individual end-users, questionnaires of the general user group/community, and observations of end-users actually interacting with a particular system. In large software development projects these techniques are often inefficient because they lead to poor communication among project members which requires a lengthy resolution process [CN91].

Our elicitation technique in terms of direct contact with the end-users will be to ask a group of users to list the processes that 'happen' when they interact with the information system and the 'things' that those 'happenings' require. For example, we will ask users to list things (=data) and happenings (=activities).

As previously stated, we need to elicit the following 'list' of information: reads, writes, transactions, frequency of transactions, locality of transaction request. The data will map to the reads and writes to the data encyclopedia. The activities will map to the transactions. No mapping (or translation) between the elicited frequency and locality information to the model representation will be necessary.

3.1.1 Group Requirements Determination

Traditionally, individual analysts have been responsible for eliciting requirements from users, synthesizing the information acquired, and then modeling or developing some representation of the system requirements. This model is typically based on the analysts own understanding of the requirements after the analyst has had a chance to synthesize the information. The users only involvement throughout the requirements determination process was as an information source. Unfortunately, when an analyst creates a model in this fashion, the structure of the model is usually understandable only to the analyst [RS77]. A model jointly developed by both the analyst and a

36

group of end-users provides a model that is easily understood and readily accepted by both parties.

In group requirements determination, analysts, developers and end-users are represented and included in the process. Researchers at the University of Arizona have advocated increased user involvement in the systems development process, indicating that this would contribute to the development of better systems. According to Hayes, user participation enables the creation of a 'better' model than that created solely by the analyst and increases the probability of implementation success [Hay91]. Electronic Meeting Systems (EMS) technology seems to be the most efficient medium for supporting this collaboration between end-users and analysts.

3.1.2 Enterprise Analyzer

The Enterprise Analyzer (EA) is a set of "tools, facilities, and processes that blends aspects of both Computer Aided Software Engineering (CASE) and the University of Arizona's GroupSystems Electronic Meeting System (EMS)" [DDH91]. The EA approach aids groups in the design of information systems by providing electronic support for requirements elicitation. EA offers groups of experts, end-users, and analysts a means to "collectively describe current procedures and identify critical elements of the proposed system" [HDD90].

EA is a mechanism by which you involve the user in the requirements analysis process in such a way as to have it culminate in a model or representation that adequately represents the users requirements. EA works toward integrating elicitation and modelling. The EA Methodology is tailored to groups working on specific projects. It is both an evolutionary and iterative process. We should not think of EA as a tool but rather as part of the process.

Typically, in an EA session, a large group of participants (end-users) is broken up into groups of three or four, with representatives from different domains. Each subgroup then tackles a different module (or subsystem). By having each subgroup be responsible for a different module, the problem domain is decomposed into subproblems. This enables us to eventually reach an understanding of the global problem.

A critical part of the enterprise analysis process is the training of participants. Participants must be trained to think of objects and relationships in general rather than as unique entities. And in our case, in terms of transactions, reads, writes, locality of transactions, and frequency of transactions.

Disciplined and coordinated teamwork is a must. Teamwork implies continuous and effective communication amongst all of the participants. Capturing the elicited information and involving participants in the process will result in clear, complete and correct results.

The EA architecture is a combination of support-team members, (users from client organization and analysts), computer hardware, and software tools for documenting and analyzing the activities of an enterprise for the purposes of information systems development and re-engineering.

3.1.3 Model Instantiation

In order to instantiate the demand model for the information services required by the users, we will employ an elicitation strategy using Enterprise Analyzer. This strategy relies on a transaction based orientation. For instance, the first step in the elicitation process will be to identify all possible transaction requests (current and future). After the transaction activities have been identified, all reads and writes associated with each transaction (i.e. the data that a particular transaction depends on and effects) are determined. Subsequently, the frequency of each transaction and the locality of the transaction requests are identified. This process is obviously iterative and thus we are not requiring

that the process be followed expressly in this order.

EA not only provides electronic support for this elicitation, but it adds structure to the requirements determination process. EA provides an efficient environment for coalescing the information for the transactions and automatically generates a model representation for each transaction. The models are in the form of a modified ICOM representation (see section 3.2).

It is important to note that this is an elicitation processes not an analysis process. The main concern at this point is to try to get as much data as possible and worry about establishing the relationships later.

A model is not fully instantiated until some analysis is performed by the analyst. The analyst is responsible for transforming the representation determined by the users based on experiential heuristics (this transformation is described in section 3.3). EA simplifies the process of gathering the information we want and provides a mechanism for tracing/tracking requirements. The elicitation results can be directly fed into the actual demand model transformation process.

3.2 Requirements Representation

A representation technique should display the following characteristics [Davis 90]:

o Facilitate communication (via an easy-to-understand lang)

o Provide a means of defining the system boundary

o Provide a means of defining partitions, abstraction, &

 projections

o Encourage the analyst to think & document in terms of the

 problem as opposed to the solution

o Allow for opposing alternative but alert the analyst to

 their presence

o Make it easy for analysts to modify the knowledge

 structure

These characteristics will serve as a basis for our proposed representation.

3.2.1 Representation Framework

Structured Analysis and Design Technique (SADT) TM was developed by Doug Ross in the early 1970s at SofTech Inc. [RS77]. SADT is a methodology for developing a "clear-cut understanding" of a problem, "documenting that understanding, and then communicating it to others" [RS85]. This methodology uses "a nonambiguous graphical notation in which natural language is embedded." This notation facilitates projection, abstraction and partitioning [Dav90].

SADT is used to construct a model for a particular problem. The model represents a hierarchy of diagrams. The graphic language consists of boxes with four sides. Each side represents one of the following: input, control, output, or mechanism. Basically, the inputs are transformed to outputs under constraint of the control [RS85].

SADT provides us with a framework within which to instantiate a demand model. In the next section, we present certain modifications to the standard SADT model representation.

3.2.2 SADT/ICOM

SADT has proven to be an efficient technique for constructing a model of a problem. Structured Analysis provides a model that describes the functional reality (i.e. what something is and what it does) of a system. SADT consists of graphical language which results in ICOM (Inputs Controls Outputs Mechanisms) models.

Traditional ICOM models include the following graphical components

(see Figure 3.2.1):

Boxes - represent activities

Arrows - represent real objects or information needed by

or produced by an activity

Inputs - data needed to perform an activity

- WHAT is done by activity

- Arrow going into left side of box

Outputs - data created when activity is performed

- WHAT is done by activity

- Arrow going to out right side of box

Controls - conditions or circumstances that govern

transformation of input to output

- WHY activity is done

- Arrow going into bottom side of box

Mechanism - the person or device that carries out the

activity

- means by which an activity is performed

- HOW activity is done

- Arrow going out of top side of box

- a downward-pointing arrow indicates a processor

that completely performs the function of the box.

The arrows connecting the output of one box to the input of another represents a data constraint (not a data flow).
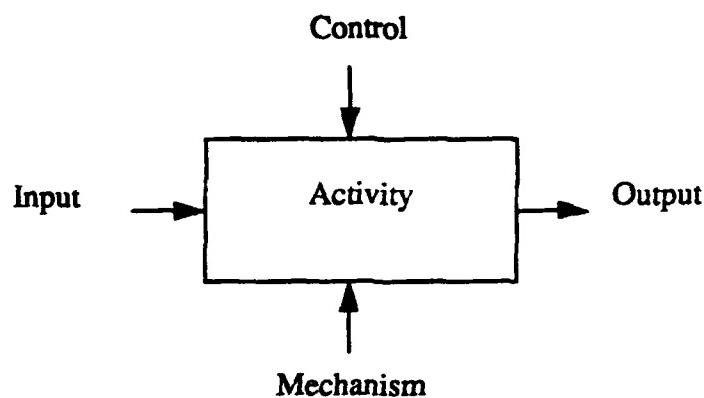
Figure 3.2.1 Standard ICOM Model

In order to properly represent the information services required by the users in the ICAT demand model, we present the following modifications to the ICOM notation. Inputs will be referred to as Reads, Controls will be referred to as Frequencies, Outputs will be referred to as Writes, Mechanisms will be referred to as Locations, and finally, Activities will be referred to as Transactions.

The modified ICOM model includes the following graphical components
(see Figure 3.2.2):

RFWM - Read Frequency Write Location (modified ICOM for ICAT)

Reads - <read>

- Arrow going into left side of box

Frequency - average transactions per day

- HOW OFTEN activity is done

- Arrow into top of box

Writes -

- Arrow going to out right side of box

Location - location of transaction request

- WHERE activity is invoked

- Arrow going into bottom side of box
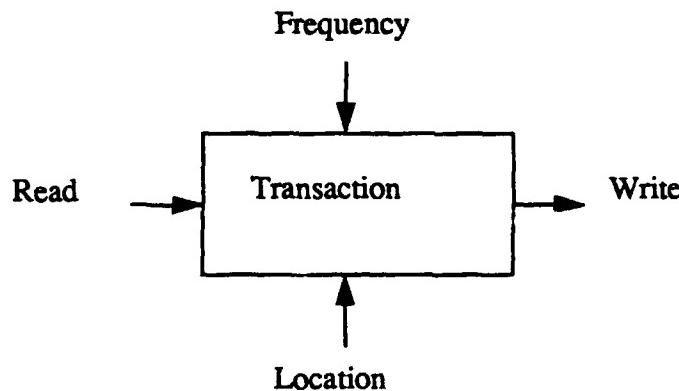
Figure 3.2.2 Modified ICOM model for ICAT Demand model.

Enterprise Analyzer automatically generates ICOM models. We propose the above modifications in order to effectively apply the existing EA support environment to the demand model information requirements.

3.3 Transformation from Requirements to Demand Model

The transformation from the <modified ICOM/SADT> representation to the information services demand model is simple. All the information required for the demand model is present in the <modified ICOM/SADT> representation.

In, fact, we anticipate that the EA tool could be modified (or a post processor added) to generate information services demand models explicitly.

The technique for realizing the demand model is as follows. First we can view the user based reads and writes as redundant and discard them. The rationale for this is that, as previously stated, the user interface (and thus his/her interactions) are part of the entry layer. The demands on the entry layer are independent. Thus we are only interested in the demands (reads/writes) on the data encyclopedia. The process of eliminating the entry layer inputs and outputs is a mental process that is run by the analyst based on experiential heuristics. The resultant set of information is thus, reads and writes to the data encyclopedia.

4.0 Sample Model

The following simplistic view of induction of a soldier is used as an example of instantiation of the model. The soldier has associated with her/him a social security number, and medical results. The output of the induction transaction is a set of orders and the creation of a personnel file. In this example, the social security number is a write to the data encyclopedia and the medical results are a write to the data encyclopedia. We are not concerned with the entry level aspect of this transaction, since that is merely a manual or other type of user input that translates to the data encyclopedia reads/writes. The outputs consist of creation of a personnel file and the printing of a set of orders. Again, we are not concerned with the entry layer component (the printing of orders), but only the data encyclopedia aspect (the creation and thus write, and possibly read of the personnel file). Thus, the induction instantiation (in our simple example), can be viewed

41

as a single transaction (and the associated frequency can be specified as an average, as well as peak), with a set of data encyclopedia reads/writes. Finally, the Location is unspecified. As was previously mentioned, the unspecified attribute allows for the induction to occur at many locations.

Frequency

SSN

Induct Soldier

Orders

Medical Results

Personnel File

Locality (who/where)
"unspecified"
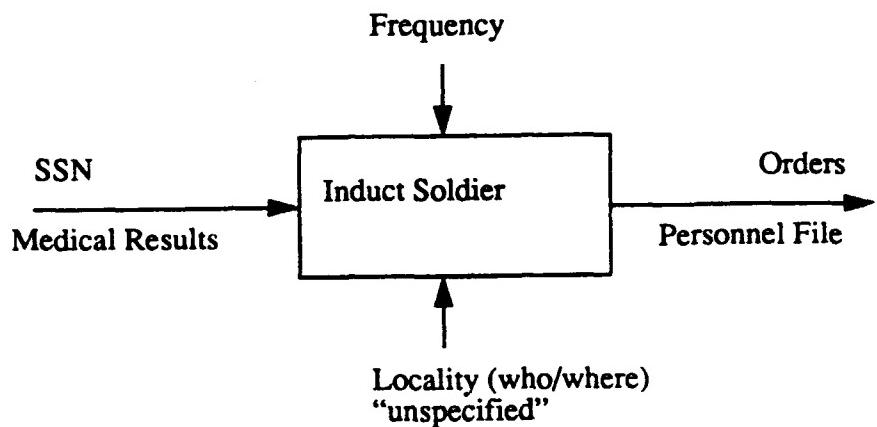
Figure 4.1. Sample Model

References

[CN91] Minder Chen, Jay F. Nunamaker, Jr. The Architecture and Design of a Collaborative Environment for Systems Definition. DATABASE, Winter/Spring 1991.

[CM90] Richard H. Cobb and Harlan D. Mills, "Engineering Software under Statistical Quality Control," IEEE Software, November 1987, pp. 44-54.

[DDH91] Robert Daniels, Alan Dennis, Glenda Hayes, J. Nunamaker, Jr.,Joseph Valacich. Enterprise Analyzer: Electronic Support for Group Requirements Elicitation. 1991.

[Dav90] Alan M. Davis. Software Requirements: Analysis and Specification. Prentice-Hall. 1992.
[Hay91] Glenda Hayes. Group Matrix Tool: A Collaborative Modelling Tool. Doctoral Dissertation. University of Arizona. 1991.

[HDD90] Glenda Hayes, Alan R. Dennis, Robert M. Daniels. Jr., V. Ramesh, J.F. Nunamaker, Jr., Doug Vogel, Joseph Valacich. Enterprise Analyzer: Electronic Support for the System Design Team. 1990.

[RS77] Douglas T. Ross and Kenneth E. Schoman, Jr. Structured Analysis for Requirements Definition. IEEE Transactions on Software Engineering, SE-3(1), January, 1977.
[RS85] Dogulas T. Ross. Applications and Extensions of SADT. IEEE Computer, 18(4), April 1985.

THIS PAGE WAS INTENTIONALY LEFT BLANK

RE-ENGINEERING AND TRANSITION

THIS PAGE WAS INTENTIONALY LEFT BLANK

ISA 97-Compliant Architecture Testbed (ICAT) Project Tools and Database Transition Subtasks

Spencer Rugaber
Bret Johnson
Gary Pardun

Software Research Center
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

ABSTRACT: This document is part of the final report of the ISA 97-Compliant Architecture Testbed (ICAT) project. In particular, it summarizes the work on the fourth task, which is concerned with the transition of Army management information systems from a mainframe, flat file, third generation environment to a distributed, open systems environment using workstations, networks, relational database management systems, and SQL. This task, in turn, has two parts. The first involves the selection and acquisition of tools to help the transition process. The second part describes a variety of strategies that might be applied depending on circumstances. The decision on which strategy to apply is described in terms of a set of criteria. A skeleton decision structure is described that allows the criteria to be applied.

1.0 Background And Motivation

Several faculty members of the College of Computing are involved in the ISA 97-Compliant Architecture Testbed (ICAT) Project, sponsored by AIRMICS. The Performance Work Statement (PWS)[3] for the project includes the following paragraph describing the project's background and objectives.

In response to the current information requirements (*standards-compliant systems*) and strategies (open systems and decentralization of data processing) of the Army, AIRMICS has proposed the "ISA 97 Conceptual Architecture Model" (ICAM) as an information system architecture reference model for building an information system using standardized interfaces and components.

The project consists of four major tasks directed at prototyping ICAM. The first two involve refinement and implementation of ICAM into a testbed called ICAT. The third major task involves modeling the demand for information services provided by ICAM. The final major task involves the

47

transition from the current information services environment into one based on ICAM. The present document describes work directed at the accomplishing the first two subtasks of the transition task (the final major task) of the ICAT project.

The transition task of ICAT involves three subtasks. The third concern transition to the GOSIP communication protocols and will not be discussed further here. The other two subtasks involve the transition of information systems to the ICAM environment. The first subtask is concerned with tool evaluation and acquisition and is described as follows in the PWS:

> The contractor shall identify CASE and reverse-
>
> engineering tools necessary for developing new applica-
>
> tions and transiting current applications. Subject to
>
> the approval of the government, the contractor shall
>
> acquire these tools.

The second subtask, concerned with strategies for transiting information systems, is described as follows in the PWS:

> The contractor shall recommend the least effort and
>
> most cost effective strategy for the transition of
>
> databases (from flat files and navigational databases
>
> to distributed relational databases) and the transition
>
> of application programs (from 3GL to 4GL).

2.0 Case And Reverse Engineering Tools Subtask

2.1 Goals

The major goals of the ICAM are to promote the use of open-systems and decentralized data processing for Army management information systems. For the purposes of the ICAT project, this implies the use of the Unix operating system on SUN Sparc workstations running a relation database management system (RDBMS). Typically, these information systems now run on an IBM mainframe, are written in COBOL, and make use of flat files to hold their data.

The transition from systems based on flat files to systems that make use of an RDBMS requires a high-level understanding of the application and how it manipulates its data. The process of obtaining that understanding is called reverse engineering. Chikofsky and Cross[6] give the following definition. "Reverse engineering is the process of analyzing a subject system to identify the system's components a:..! their interrelationships and create representations of the system in another form or at a higher level of abstraction."

Reverse engineering is a labor intensive process. It is therefore desirable to automate it to the extent possible. Part of the aim of this subtask is to investigate tools that can facilitate this process. To accomplish these end, the following tools have been investigated.

> o   Unix COBOL compilers: COBOL compilers and run-
>
> time systems capable of targeting Unix worksta-
>
> tions.

o   Relational database management systems: RDBMSs
    capable of running on a workstation and being
    accessed across a network.

o   Computer-Aided Software Engineering (CASE) tech-
    nology: integrated tool sets capable of
    representing high-level designs of information
    systems and aiding the generation of enhanced pro-
    grams.

o   Program analysis tools: to facilitate the
    transformation of flat file COBOL programs to ver-
    sions capable of interacting with an RDBMS.

o   Other technology to support the transition pro-
    cess.

2.2 Tools Identified And Acquired

2.2.1 Open Systems Cobol Compiler Technology

Several vendors advertise COBOL compilers targeted at Unix workstations. We selected and pur-
chased the COBOL tools from Liant Software. In particular, we purchased and installed the RM/
COBOL*85 compiler and development environment. This includes the compiler itself, the run-
time environment that supports execution, and several integrated development tools.

2.2.2 Relational Database Management System

We chose to take advantage of the existing Georgia Tech site license for the Oracle RDBMS run-
ning on the Sparc workstation. In particular, we installed the SQL*Plus interactive interface tool,
the SQL*Forms forms interface, and the Oracle SQL*ReportWriter, as well as the database server
itself.

2.2.3 Case Tools

We chose to take advantage of an existing Georgia Tech site license. In this case, the license was
for the Software Through Pictures (STP) CASE tool from Interactive Development Environments,
Inc. (IDE).

2.2.4 Program Analysis Tools

In order to support the transformation of flat file COBOL programs to programs that access an
RDBMS as cost effectively as possible, it is desirable to automate as much as possible the analysis
and adaptation process. To do this, we made use of a publicly available program analysis tool called
NewYacc[11,13] with which we had previous experience. NewYacc augments a grammatical
description of the language in which the programs that it is analyzing are written. In this case, the
language is COBOL. In order to obtain a grammar for COBOL, we purchased the PCYACC tool
from Abraxas Software. Unfortunately, despite their advertisements of a Unix version of this gram-
mar and the associated tools, they could only supply a version for the PS/2.

Other Technology To Support The Transition Process

It should come as no surprise that there are some vendors who supply tools to support the kinds of transition with which this project is concerned. We were able to obtain an attractive one called RM/plusDB, a RDBMS bridging package, also from Liant Software.

2.3 Status And Evaluation

The next section describes how the acquired tools were used during this project. This subsection summarizes our experience acquiring, installing, and testing the tools.

- o COBOL compiler: our guess is that this tool has itself been recently converted to run in a Unix environment and that the transition was not smooth. We had difficulty getting the compiler to run in our environment, which includes distributed file services. In fact we had to upgrade our version of the operating system in order to get consistent compilations. Once we did this, we had no further trouble.

- o COBOL run-time system: this seems to work, but has not been extensively tested due to the late arrival of the test data.

- o COBOL development tools: these ran satisfactorily.

  Relational database management system: the Oracle software fills three SUN cartridge tapes! So installation was quite tedious. Once this was accomplished, it worked satisfactorily.

- o CASE Tools: STP was acquired, installed, and tested without difficulty. It ran satisfactorily.

- o NewYacc: was already installed. It ran satisfactorily although it places some minor constraints on the names for tokens that can be used in the grammar description file.

- o PCYACC: although this software was advertized to run on Unix, and the vendor billed us for a Unix version, it became apparent that what we had actually received was a PC version. A telephone conversation to the vendor revealed that they have

not yet released their Unix product. They suggested that we take the grammar from the PC version sion and run it using the standard Unix tools (YACC and LEX). They felt that it should run without difficulty. After considerable difficulty, we did get it to run using an adapted version of the Berkeley release of YACC. We have since gotten them to adjust the price.

o   RM/plusDB: this product claims to invisibly support the replacement of flat files by RDBMS tables. However, the version that they sent us did not support Oracle. The Oracle version is about to be released, and the vendor has agreed to send us a no-cost upgrade.

Overall, it is fair to say that our experience with the vendor-supplied Unix tools was somewhat frustrating. We had to go to a considerable amount of effort to get working versions of the tools that we needed. The situation should gradually improve as more vendors target the Unix marketplace.

3.0 Database And Information Systems Transition Subtask

3.1 Goals

This Database and Information Systems Transition subtask requires the selection of a strategy for the transition of systems from flat files to RDBMSs and from the COBOL programming language to fourth generation languages. It is apparent, however, from the range of application systems that no single strategy will suffice. Therefore, section 3.2 will discuss a variety of strategies that we explored. Sections 3.3 and 3.4 propose criteria by use of which an appropriate strategy can be applied to a specific situation.

3.2 Strategy Characterization And Experience

A variety of strategies can potentially be applied to move a management information system to a distributed opensystems environment. They range from doing nothing (for systems where the cost of transition outweighs any benefits) to a full rewrite in the new environment. This subsection begins by describing the spectrum of existing systems. Then a sample system that was used for our experimentation is described. Finally, the range of strategies that we investigated is presented.

3.2.1 Army Management Information Systems

Two categories of Army management information systems are of concern to this project. The first category includes the STANDARD ARMY MANAGEMENT INFORMATION SYSTEMS (STAMIS's). Of the 187 STAMIS's listed in the STAMOD database, 118 were written at least partially in COBOL.* When all systems and variants are considered, the average size of a STAMIS is 206 thousand lines of code. Each such system is broken up into separate programs. The average number

51

of programs per system is 161. Of the original systems, 78 already access a database of some sort. These databases range from large-scale third generation systems like DMS1100 to PC record managers like DBase. Some systems already access RDBMSs such as Oracle or Informix. Six systems in the STAMOD database list their DBMS as "TBD", presumably to indicate that the system is under construction, and the DBMS has not yet been decided.

The other category of Army management information system is called an INSTALLATION SUPPORT MODULE (ISM). These systems have not been standardized and may, in fact, be relevant only to a single Army installation. An effort is underway to use the ACE prototyping environment to move these systems toward a distributed open-systems environment. The limited time allowed for the project did not permit them to be included in the study.

3.2.2 A Typical Army Management Information System

For the purposes of this project, we needed to explore the difficulties that arise when actually transiting a STAMIS. To do this, we used the Installation Materiel Condition Status Reporting System (IMCSRS)[1]. This STAMIS consists of approximately 10,000 lines of COBOL code, broken into 15 programs. We had become familiar with this system during an earlier project[12] that ultimately lead to the replacement of IMCSRS by a version written in Ada[7].

IMCSRS is smaller than a typical STAMIS, but it performs typical functions. It is responsible for using input transactions to update a master file and then producing a variety of reports describing the status of Army materiel. Most of this functionality could ultimately be replaced by RDBMS activities expressed using a fourth generation

* It should be noted that this database includes multiple occurrences of systems with the same name. The occurrences have different properties; so the presumption is that they represent significant variants. It is unclear how up-to-date the STAMOD database is language (4GL).

3.2.3 Strategies Investigated

A variety of approaches are conceivable when considering the transition of an information system to a distributed open-systems environment. This subsection lists those at which we have looked. They are organized roughly from those requiring the least to the most effort to effect.

The strategies that are described in this section are comparable in the sense that each has a costs and benefits. For any given situation, the costs and benefits for the various strategies must be compared to select the most cost-effective approach. The next subsection describes questions that can be asked in order to make these judgements. This subsection merely describes the strategies and our experiences with them.

3.2.3.1 As Is Strategy

The base line against which the other strategies must be measured is the strategy of doing nothing. In this case, there is no real benefit, and the cost is fairly well understood. This strategy may be appropriate if it is known that the STAMIS is going to be replaced or phased out. It may also be applicable if the STAMIS is used only infrequently by a single site. In this case, there is little value in supporting distributed access.

3.2.3.2 SQL Access To Flat Files Strategy

Another strategy that occurred to us leaves both the COBOL program and the data files untouched. This strategy involves the development or acquisition of a tool that provides SQL access to flat files. This strategy allows new queries to be written strictly in SQL without requiring COBOL code

alteration. It also supports the incremental replacement of COBOL functionality by SQL queries. Another advantage of this approach is that it provides another tool in obtaining a hybrid environment, where some programs are written in COBOL and some in SQL and some data files are flat files and some are relational database tables. We did not investigate this option experimentally.

3.2.3.3 Direct Stamis Porting Strategy

Another strategy is to simply port the COBOL system from the mainframe environment onto a Unix workstation without adding any new functionality. We attempted to do this with IMCSRS. Unfortunately, IMCSRS is written in an older dialect of COBOL, COBOL 68. The compiler which we have obtained (as well as the PCYACC program analysis tool) is capable of compiling the COBOL 74 and COBOL 85 dialects.

In order to pursue this strategy, we had to convert the COBOL code by hand in order to compile it. This was time consuming but relatively straightforward. Appendix A lists the specific conversions that had to be made. The STAMOD database does not include information about the version of a programming language that was used to build an information system, so we have no way of knowing how frequently such conversions will be required.

After the code was converted, it successfully compiled on the Unix workstation. In order to test the accuracy of the conversion, however, we needed to obtain sample IMCSRS test data. We were not able to obtain this until mid-December. Since it has been received, we have been able to run several of the programs and successfully obtain reports. Examples are included in Appendix E. This gives us some indication that the COBOL run-time system is working.

Another factor to consider is the conversion of system data. Issues such as word size, byte order, and character set may cause difficulties in applying this strategy. The unavailability of the test data prevented us from looking into these issues more fully. When the test data arrived, it did require conversion. Specifically, the data had to be converted into newline-terminated Unix format. We did not have any mechanism for determining whether any word size differences, such as might exist between a 36-bit mainframe and a 32-bit workstation, would lead to loss of precision.

The direct porting strategy may be desirable when it is advantageous to move a system off of a mainframe. This may happen if execution costs are significantly reduced by using a workstation or if the network access provided on the workstation increases the customer base and timeliness of the application's reports. It may also be applicable as an interim step to some of the strategies described below.

3.2.3.4 Transparent Layered Conversion Strategy

Some vendors are already addressing the software transition problem. For example, Liant Software, vendors of the COBOL compiler that we purchased, sells a tool called RM/plusDB. Its purpose is to provide a transparent mechanism so that existing COBOL programs can access RDBMSs without having to be altered. RM/plusDB provides an extended run-time environment and a server. The run-time extensions are invoked when a COBOL statement attempts to do I/O to a flat file. The run-time routines, instead, convert the I/O request to an access to the RDBMS. The server transfers these requests to the RDBMS and returns any data passed back from the RDBMS.

The steps required to implement this strategy on an information system are the following:

o   Extract and annotate the File Description state-

    ments (FDs) from the COBOL code. The annotations

appear as COBOL comments.

o   Using an RM/plusDB utility, create a data diction-
    ary describing each of the files.

o   Using another utility, convert the data dictionary
    into database table definitions. Note that this
    step is dependent on the particular target RDBMS.

o   Use the data loading capabilities of the RDBMS to
    load the flat file data into the RDBMS.

o   Relink the application program with the new run-
    time environment.

This strategy provides some of the sought-for benefits. All of the advantages of DBMSs over flat files such as data security and integrity are available. Also, enhancements are facilitated--new reports can be easily constructed using the 4GL capabilities of the RDBMS. Because the code is not altered, the cost is low. This strategy can also be applied as an interim step to those described below.

Unfortunately, the RM/plusDB tool does not yet support the Oracle RDBMS that we were target-ing. We were able to accomplish the first two of the steps described above, but the appropriate server is not available to us to allow testing. They are nearly ready to release their Oracle version, and we have arranged to obtain a no-cost upgrade that supports Oracle.

3.2.3.5 Code Layering Strategy

The previous strategies have avoided altering the source code of an application system. Sometimes, however, the benefits of direct intervention are warranted. Most database vendors provide a mech-anism for directly placing Structured Query Language (SQL) statements into source code. This is typically accomplished using either a preprocessor or through direct library calls. We did not try any specific experiments with this strategy, but some of the observations related in the next section are relevant here.

3.2.3.6 Code Replacement Strategy

4GL programs are smaller and more maintainable than are programs written in third generation languages like COBOL. These benefits can be a strong inducement to replace parts or all of an application program by 4GL programs. Specifically, 4GLs support the construction of reports and the satisfying of relatively small queries that do not require a great deal of computation. Also, depending on the specific interactions involved, the VIEW and JOIN capabilities of an RDBMS can replace some complicated computations involving multiple files. In addition, some 4GLs pro-vide declarative data validation mechanisms that can further reduce code size.

We conducted an experiment, taking one of the IMCSRS programs and converting it to SQL. In order to accomplish this, the original COBOL program had to be understood. Traditionally, this would be accomplished informally--trying to understand the COBOL program in terms of the tar-get SQL functionality. Although this was the course that we took, our recommendation would be to make this re-engineering process more systematic. In particular, an explicit reverse engineering step is required in which a representation of the program's data processing requirements is con-

structed. Such a procedure targeted specifically at information systems is described in Batini's book[4]. Another approach, aimed somewhat more at the functional requirements than at data modeling is described in the technical report by Kamper and Rugaber[9].

The experiment was successful in the sense that SQL reports were generated from the Oracle database. The success is somewhat tempered, however, by the lack of actual test data with which to compare results. Also, it is important to realize that we intentionally selected a program whose purpose was to generate a report, so it is not surprising that we were able to do so. Perhaps the most accurate conclusion to draw is that we now have at least one instance where this particular strategy has succeeded. The experiment was conducted by one of the authors (Johnson). His description of the experience is included as Appendix B.

This strategy appears to be applicable in an environment where incremental transition from COBOL to SQL is desirable. This may be warranted where the programmers are receiving on-the-job training and need small examples. The feedback obtained from comparing the existing results with that obtained from SQL can serve to validate the conversion. If the program being converted is convoluted and difficult to understand, incrementally removing segments responsible for relatively self-contained tasks, such as report generation, can reduce the size of the remaining COBOL code to the point where direct comprehension is more feasible.

3.2.3.7 RDBMs Conversion By Reengineering Strategy

Program evolution without benefit of a high-level representation of functionality and structure presents risks in terms of quality. The process of reverse engineering existing software yields such a representation that can then be used as a basis for enhancements.

The benefits of such an approach are obvious; the costs are, however, difficult to measure. One factor that needs to be understood is that reverse engineering requires a significant commitment in time and effort. Some discussions of mechanisms for partially automating the process are described in the next strategy subsection.

The manual re-engineering strategy is indicated in situations where the existing code will continue to be used extensively for the foreseeable future. Maintenance activities that require modification of existing code (versus simply adding new modules) can also help justify the expense of reverse engineering. Reverse engineering does not have to be applied to an entire system. The log in Appendix B describes its informal use on one component of IMCSRS. In this case, however, previous reverse engineering work significantly facilitated the process. Even if only a part of a system is being reverse engineered, there is still a need for the reverse engineer to understand the context of the component relative to the entire system. Thus, in situations where resources such as accurate documentation or experienced maintenance personal exist, partial reverse engineering may by more feasible.

3.2.3.8 Automatic Reverse Engineering Strategy

Because of the expense involved in reverse engineering, it is desirable to automate as much as possible the steps involved. Unfortunately, the state of the art is such that few tools exist and those that do are capable of describing only surface features of an existing system.

The strategy of automatic reverse engineering involves extracting features from existing programs and translating them into a standard design representation. We performed an experiment to explore the feasibility of this approach. Specifically, we wanted to know whether we could automatically extract information from a COBOL program describing the structure of the files that it uses. There

are three components of the effort: program analysis, transformation of design information, and design representation and display.

3.2.3.8.1 Program Analysis

COBOL programs are highly structured descriptions of computations and data. The programs represent the culmination of a series of design decisions that transform an initial specification into a final program. Moreover, even after a program is delivered, it undergoes subsequent changes for the purposes of removing defects, adding enhancements, and adapting to changing environmental constraints.

In order to construct a high-level design description of a software system, the design decisions that went into its construction and maintenance must be reconstructed. This is accomplished by a systematic analysis of the program text, simultaneously constructing a description of the application domain and procedures that the program models. The analysis can be performed manually, but the process is labor intensive and therefore costly in time and resources. It is desirable to replace as much as possible of the manual effort by the use of automated tools.

Because of the structured nature of COBOL, any analysis must be based on a grammatical description of that language, i. e. a COBOL grammar. We have previously made use of a grammar-based tool called NewYacc[11,13]. NewYacc is a preprocessor to the Unix yacc tool. Yacc takes as input a grammatical description of a programming language and produces as output a parser suitable for parsing programs written in that language. To yacc's rules, NewYacc adds two features. First of all, NewYacc augments the constructed yacc parser so that when it runs it retains a description of the structure of the program that it is parsing. This description is a parse tree (also called a concrete syntax tree), and normally the yacc-constructed parser builds it implicitly and then throws it away when parsing is completed.

This extension to yacc is complemented by the second additional feature. Normally yacc allows its users to describe not only the structure of the language whose programs will be parsed, but also semantic rules describing how the program will be translated or interpreted. Thus, yacc is often used to construct a parser that serves as the first step of a compiler. To this power, NewYacc adds another kind of grammar annotation. The annotations describe rules that can be applied during traversals of the parse tree.

The rules are of two forms, both of potential use during reverse engineering. The first kind of rule describes a SELECTIVE TRAVERSAL. These rules indicate which program features should be extracted from the parse tree during the traversal. In the case of IMCSRS, we are interested in extracting the File Description statements (FD's). These can be used for several purposes to automate parts of the other strategies. For example, in the Transparent Layered Conversion strategy, it was necessary to use the FDs in order to build a description of the tables in the RDBMS that hold the input and output data. Also, in the RDBMS Conversion by Re-engineering strategy, we would like to construct a high-level representation of the data manipulation requirements of the program.

Entity-Relationship diagrams[5] (E-R diagrams) are an appropriate medium for this task. In these diagrams, each input and output file corresponds to one entity. To automate the process, we would like to extract the FDs and convert them into the description of entities in the diagram. Selective traversal easily provide this information.

The other kind of NewYacc rule supports OPEN TRAVERSALS. In this case, the rules describe how an input program can be systematically transformed. For example, it might be desirable to uni-

formly apply indentation rules to an existing programs. Also, renaming program variables so that they more clearly express their use in a program can be done in such a way that all and only those occurrences of a series of characters in the program text that actually denote the variables to be changed are affected. Occurrences of the characters in comments or string literals are ignored.

In order to use NewYacc on IMCSRS, a COBOL grammar is required. COBOL is a relatively new language in the Unix world, grammars are not readily available as they are for other languages such as C or Pascal. We were, however, able to find a vendor that supplies a version of yacc and lex (a Unix program manipulation tool used to build lexical analyzers for programming languages). This vendor is Abraxas Software and their product is called PCYACC. PCYACC is primarily intended to provide yacc and lex to the non-Unix PC world. They do advertise, however, a Unix-based version that includes grammars for a variety of languages, including COBOL.

Unfortunately, when we obtained this software we learned that, despite their claims, they still only had a PC version. They advised us, however, to transport the grammar to a Unix workstation and use it with the yacc that was there. They foresaw no difficulty with this process.

We were eventually able to accomplish this task, but it was considerably more involved than they claimed. In fact, there were several difficulties that COBOL presented. The difficulties are described in Appendix C.

Once we had a working tool, we were able to integrate NewYacc. By the end of project schedule, we had gotten to the point where NewYacc was working with COBOL, and we were beginning to construct rules for extracting COBOL constructs. We foresee no further difficulty in using this mechanism for extracting FDs.

3.2.3.8.2 Transformation Of Design Information

Our NewYacc rules are designed to analyze a COBOL program using selective traversal to extract FD information. Once the FDs are extracted they can be used by RM/plusDB as part of the Transparent Layered Conversion strategy or as a basis for one part of the Automatic Reverse Engineering strategy. Specifically, we want to be able to construct an ER diagram that describes an application's data model. In this diagram, application files will be mapped to entities. NewYacc is capable of extracting the FDs, and the Software Through Pictures (STP) CASE tool is capable of displaying the diagram. The remaining step is to transform the FD information into a form understandable by STP.

STP has an OPEN ARCHITECTURE. This means that it is extensible to users in a variety of ways. In particular, diagrams are stored using a textual representation, and the format of this representation is documented. The normal mode of diagram construction in STP is by the end user manually selecting icons and placing them in the diagram on the screen. Using the published file format, however, we were able to automatically construct diagrams based on the information extracted by NewYacc. A technical report describing this process has been written by Johnson[8].

In essence, the program has to be able to perform three tasks. It must first read in the information extracted by NewYacc and construct an internal representation of a file described by an FD. Second, the program must build up an internal description of the corresponding diagram that will be displayed. This process is called layout. Finally, the program must interpret this description and produce a textual representation readable by STP.

We have previously performed this process using a SmallTalk program that produced Structure Chart diagrams. The Structure Charts describe the calling structure (call tree) for C programs. For

this project, the program was converted from SmallTalk to C and modularized into components corresponding to the three activities listed above.As mentioned above, we did not quite get to the point where NewYacc extracted FDs from COBOL programs. However, the call tree problem is inherently more difficult than constructing a picture describing only a single entity and its associated attributes, so we foresee no fundamental difficulty with completing the tool.

3.2.3.8.3 Design Representation And Display

The diagram that is constructed by this process represents one COBOL file. Such a file correspond to an entity in an ER diagram. The entity and its attributes are obtained from the FD statements. Uses of such a diagram include visualization of the file's structure, graphical manipulation using the CASE tool's editing features, and automatic schema generation.

Visualization is limited by this approach in several ways. First, STP provides no mechanism for refining an ER diagram into lower level diagrams. This is not a fundamental limitation. The Batini book[4], for example, describes several ways in which complex diagrams can be abstracted. If a diagram got very crowded, its ability to promote visualization would be correspondingly reduced. The second limitation has to do with RELATIONSHIPS. An ER diagram consists of entities and relationships. In our case, the entities correspond to files and can be automatically extracted using the procedures described above. Relationships, however, are more troublesome. Relationships between an input and an output file can be arbitrarily complex, and many other relationships are only implicitly apparent in the code. We see some promise in using a technology from compiler theory called dataflow analysis to estimate such relationships. One of us (Pardun) has written a technical report for his database class proposing how this problem might be attacked[10].

Once an ER diagram has been built by the automatic procedure, it is editable just as if the diagram had been drawn initially from within the CASE environment. This provides support for future enhancements to the information system. For example, if a program enhancement involved adding new information to an input file, the corresponding ER diagram could be extracted and edited. Then the CASE tool's template generation mechanism could be used to automatically produce a new version of the FD statement describing the file. In a sense, then, maintenance has been moved from a code editing activity to a conceptual alteration expressed graphically.

STP has another feature that allows it to automatically construct table descriptions for a variety of database systems. In particular, it can construct SQL CREATE statements for Oracle. To explore this feature, we tried a small experiment. One of us (Johnson) took an FD from one of IMCSRS's program (P13AGU) and manually constructed an ER diagram for it using STP. This is the process described above that we believe is automatable. His description of the process and the difficulties that he encountered are described in Appendix D. This process supports a transition strategy where an existing COBOL input file is automatically converted to an Oracle relation. Of course, we have only looked at part of the problem; it also seems possible to automatically generate the statements to the RDBMS's data loader utility to load the file's contents into the RDBMS.

3.2.3.9 From Scratch Rewrite Strategy

A final strategy needs to be mentioned for reasons of completeness. Under some circumstances, it may be desirable to throw out the existing program entirely and to rebuild from scratch, including new requirements gathering. This situation may obtain when the old system needs to be significantly modified and its complexity is severe enough that the cost of re-engineering is outweighed by the costs (and risks) of initial development.

3.2.4 Summary Of Strategies

The following table lists the strategies described in this section.

Strategy
As-Is- Strategy SQL Access to Flat Files Strategy Direct STAMIS Porting Strategy Transparent Layered Conversion Strategy Code Layering Strategy Code Replacement Strategy RDBMS Conversion by Re-engineering Strategy Automatic Reverse Engineering Strategy From-Scratch Rewrite Strategy

3.3 Decision Criteria

The previous section describes a wide variety of strategies, no one of which is suitable for dealing with all situations. In order to determine which strategy is appropriate in a given situation, those factors that can effect the costs and benefits of applying the strategy should be weighed. These factors are called decision criteria, and this subsection lists and defines them. The next subsection proposes a mechanism whereby the criteria can be organized into a structure that will facilitate the decision making process.

3.3.1 Factors Related To Usage Of The Existing System

o   Usage profile and availability: How many users does the

system currently have? How are these distributed

topologically (are they logged into the mainframe, do

they submit batch jobs, or are run requests handled

manually)? How frequently does a given user make use

of the application? In what different ways are the

application used (what is the ratio of data updates to

reports produced)? How frequently is each such use

made? What is the physical process by which a use of

the application is currently made (data entry, valida-

tion handled separately; manual or electronic distribu-

tion of reports)? How many different sites use the

existing system?

o   Expected lifetime: What is the expected lifetime of

59

the existing application? Is usage growing or shrink-
ing?

o   Current execution costs: How much does it currently
cost to execute the program in terms of machine and
human resources? How does this cost vary across the
various types of uses?

o   Ownership and control: Are there political factors
that would impede the reduction in information control
that comes from distributed access?

o   Administration: Are there administrative procedures
that would be difficult to provide in a distributed
environment? What are the costs in transforming these
procedures?

o   Interoperation: Do other applications depend directly
on the data produced by this application (master file,
report files, exception files)? Does this application
depend on the products of other applications?

3.3.2 Factors Related To The Structures And Functionality Of The Existing System

o   Current architecture: How amenable is the current
architecture to the client/server model? Is the appli-
cation primarily batch or interactive?

o   Hardware configuration considerations - type and
resource availability: What external resources and
connections does the application require? How exten-
sively are these used?

o   Software configuration considerations: Does the exist-
ing system make use of non-portable operating system
capabilities? Does the existing system interface to
other existing systems?

o   Reports: Does the existing system write reports? If
so, how separable is the computational functionality
from the report construction functionality? Are there
reports · could be replaced by SQL queries? Are
there re;   :s that could be replaced by reports con-

60

structed by the RDBMS report writer capability?

o   Other RDBMS features: Does the current application do
    significant data validation that could be replaced by
    the data validation features of the RDBMS? Could the
    current application make effective use of advanced
    RDBMS operations like views and joins?

3.3.3 Factors Related To Expected Usage Of The Transited System

o   Increased usage: What is the expected increase in
    usage of the system due to networked availability?
    What is the expected change in usage (e. g. from batch
    to interactive) promoted by distributed access?

o   DBMS functions: Can the application take advantage of
    DBMS capabilities such as security and integrity?

o   Proposed execution costs: What is the expected change
    in execution cost in terms of machine and human
    resources?

3.3.4 Factors Related To Expected Evolution Of The Transited System

o   Technical impediments: Does the existing system make
    use of a DBMS? Is it relational? Does the existing
    system make use of an older COBOL version? Are there
    portability issues related to data conversion? Can
    this application be integrated into others?

o   Maintenance requirements: How much corrective mainte-
    nance activity is there currently on the system? What
    enhancements to the system are planned? What enhance-
    ments would be facilitated by the use of an SQL inter-
    face to the data?

o   Support issues: Are there personnel available that
    have experience with the internals of the existing sys-
    tem? Is there existing documentation for the system?
    How up-to-data and accurate is it? Is sufficient fund-
    ing available for a comprehensive reverse engineering
    effort? Does this include funding to support the
    training of users in 4GLs? How feasible is incremental

conversion?

o   Standards: Is the application part of the effort to
standardize the use of data item names? How closely
does it conform to these standards?

3.4 Skeleton Decision Structure

The various factors and strategies outlined have an internal structure. That is, making one decision
may naturally lead to another question being asked? Some decisions preclude others? To summa-
rize the possibilities, the decisions can be organized into a hierarchical structure called a decision
tree. A decision tree for the transition strategies described above is given by the following. The
numbering of the questions indicates their relative placement.

1. Should the information system (IS):

   1.1. be left alone (As-Is Strategy)?

   1.2. be moved into an open system/4GL environment?

1.2. In moving to open systems, should the IS

   1.2.1 be ported as-is (Direct STAMIS Porting Stra-
   tegy)?

   1.2.2 be migrated to an RDBMS/4GL?

1.2.2A If migration is desirable, should the new ver-
sion

   1.2.2A.1 migrate the data to a RDBMS and leave the
   code alone to the extent possible, concentrating
   instead on the file data (Transparent Layered
   Conversion Strategy)?

   1.2.2A.2 replace selected features of the code
   with 4GL constructs?

   1.2.2A.3 provided the functionality with SQL
   access to flat files (SQL Access to Flat Files
   Strategy)?

1.2.2A.2 If code is going to be replaced, which combi-
nation of 4GL features should be replaced:

   1.2.2A.2.1 File access replaced by embedded SQL
   and/or library calls (Code Layering Strategy)?

   1.2.2A.2.2 Report formatting replaced by the 4GL
   report writer?

   1.2.2A.2.3 Some reports replaced by SQL or forms-

based queries?

1.2.2A.2.1 If file access is to be replaced, should it be accomplished

    1.2.2A.2.1.1 with a layering tool?

    1.2.2A.2.1.2 by replacing COBOL I/O with embedded RDBMS access?

1.2.2B If migration is desirable, should the new version

    1.2.2B.1 be developed from scratch, including requirements gathering (From-Scratch Rewrite Strategy)?

    1.2.2B.2 be the results of reverse engineering the existing version (RDBMS Conversion by Re-engineering Strategy)?

    1.2.2B.3 be incrementally migrated from the current version (Code Replacement Strategy)?

1.2.2B.2 If reverse engineering is used, should it be done

    1.2.2B.2.1 by hand?

    1.2.2B.2.2 using semi-automated tools (Automatic Reverse Engineering Strategy)?

In order to use the decision tree described above, the criteria described in the previous subsection must be interpolated. There are two substantial impediments to doing this? The first impediment involves measurement. While some of the decision criteria are quantifiable ("average cost of an execution of the application program"), many of them are qualitative ("can the application take advantage of advanced RDBMS features such as views and joins?"). In order for there to be a formal decision procedure, metrics for the qualitative factors need to be devised and validated.

The second impediment involves combining the criteria. Even if all criteria were quantitative, it would still not be meaningful to make a decision that combine the numbers from two criteria. For example, is an application currently uses a non-relational DBMS? How does this mitigate the advantages of transiting to an RDBMS?

Two possible approaches to dealing with these impediments are suggested: scenarios and case studies. A scenario is a hypothetical narrative describing a potential transition effort. It can be used to explore difficulties in applying the criteria. Scenarios have been effectively used to solicit system requirements during the early stages of software development, and their use here has the same purpose, to unearth unanticipated costs and benefits.

The second suggestion involves the use of case studies of previous transition efforts. In particular,

data concerning costs involved, difficulties that arose, and eventual benefits can serve to seed the decision structure.

3.5 Integration With The Activities

The two subtasks described in this report are part of a larger effort, the ICAT project. Moreover, related efforts exist in the Army on which this work impinges.

3.5.1 ICAT Architecture Requirements

The ICAT project involves the construction of a testbed of networked Unix workstations. There are really two environments to be considered, the transition environment and the resultant distributed information system execution environment. The development environment requires the inclusion of a Unix COBOL compiler. As mentioned above, the compiler we obtained initially had difficulty running in a distributed file system environment. Once this problem was solved, we were able to compile and run the trial system. This is no guarantee, however, that other information systems will not tax the compiler beyond its abilities to cope.

In order to support re-engineering, the development environment must also be able to support the tools that we have acquired. This includes the availability of X Windows. Not only is X Windows used by the CASE tool, but the possibility also exists of enhancing the interface to the application systems using X Windows graphics capabilities. Yet to be considered are the issues of how to design a widget set to promote the standardization and reuse of information system interface components.

Two other development requirements were mentioned earlier. They are the availability of a version of yacc that can handle the number of different reserved words found in COBOL. Both byacc (from the publicly available Berkeley software distribution) or bison (from the Free Software Foundation) may be suitable. The other requirement relates to the Oracle version of the layering tool. It must be possible to access the RDBMS through a layered interface such as that provided by Liant.

Execution also places restrictions on the environment. One prominent restriction concerns disk space. Not only does Oracle require significant space to install, but the amount of data normally presented to a mainframe application may overwhelm a typical workstation. Of course, network access and execution cycles also must be gauged before an application can be assured of adequate execution resources on the workstation.

Currently, application programs running on a mainframe are accessed through PCs or dedicated terminals. In both cases, the interface is typically character-oriented. This fits well with Oracle's forms capabilities, but the desirability of moving to the more versatile X Windows interaction environment will certainly impose terminal emulation requirements.

3.5.1.1 Data Encyclopedia Project

The Army is currently developing the ANSWER[2] system to serve as a single point of query entry for information systems. The intent is to promote standardization particularly in the area of database schema. A prototype exists, but efforts to address information systems that use flat files are currently not being pursued. If a transition effort is going to involve significant rework of existing code, it may be advantageous to simultaneous convert to Army standard naming conventions. Progress on this effort should continue to be monitored.

3.5.2 Other Efforts

Two other projects that should be monitored are the STAMIS Modernization effort being con-

ducted from Fort Belvoir and the ACE/ISM project. An Installation Support Module (ISM) is an information system that does not have STAMIS status. This may be because it is tailored to a specific installation. An effort is underway to reduce the maintenance requirements on these systems by the use of ACE. ACE is a forms-oriented prototyping environment for information systems. It can be used to generate C code from forms definitions.

4.0 Recommendations For Future Work

The ICAT project only lasted five months. Many of the experiments that we tried were necessarily curtailed by the short time frame. Moreover, problems with vendor supplied software delivery and installation also forced compromises. Finally, delayed delivery of test data reduced the amount of hands on exposure to the tools. Hence, interesting work remains to be done to take advantage of the experience that we have had so far. In addition to following through with these tasks, several other areas of exploration are apparent.

4.1 Automation

This is the most direct follow on to the current project. It involves activities related to the process of extracting information from an existing STAMIS, such as the detection of opportunities to move from 3GL code for report formatting to 4GL report writers, detection of opportunities for replacing reports by direct queries of an RDBMS, and continued work on extraction of ER diagrams from code.

In addition to extraction activities, other opportunities for automation include schema normalization and data conversion.

4.2 Representation

Fundamental to the transition process of moving from COBOL code accessing flat files to the use of an RDBMS, is the need for first constructing a high level representation of the program, such as ER diagrams. While these appear adequate for modeling the relevant file structures, a more general representation is required to model the computation and control information. We would like to look at the issues involved in such a representation. This could involve a case study of extending the models from the current project.

4.3 Case And Prototyping

An important part of the STAMIS transition process is the use of CASE tools. These become even more important when initial development is considered along with maintenance. In particular, CASE tools often include a prototyping component. The question to be looked at is how best to facilitate the movement from prototype to product, while ensure thoroughness and adherence to quality standards.

At this stage in their evolution, commercially available CASE tools feature design representation that are primarily graphical, stressing the structural aspects of a design relative the semantic/functional aspects. An alternative approach is available with some prototyping systems that stress the specification of functionality using a notation such as VDM. It is possible, for example, to automatically generate Ada code from VDM specifications. We would like to look at the question of integrating these two approaches (the structural and the functional) to best provide the benefits of both.

4.4 Process Modeling

Another area of investigation involves further modeling the STAMIS transition process. The cur-

rent project will produce strategies for transition and criteria for selection among them. This understanding might be further refined by describing it using a rule base or expert system.

One other extension involves the inclusion of cost data to the decision process. For example, a maintenance shop, when deciding whether to convert an existing STAMIS to a 4GL, should consider the amount of current maintenance activity and the cost of conversion. This would serve as a further refinement for the criteria provided by the current project.

APPENIDX A COBOL 68 CONVERSION GUIDELINES

GARY PARDUN

Although IBM mainframe COBOL is quite similar to Unix COBOL (in particular, RM/COBOL85), porting COBOL programs requires a number of source code changes to successfully compile programs on Unix systems. The following list of changes reflects the modifications we had to make in the IMCSRS application programs to compile successfully on Sun Sparc stations using RM/COBOL85.

The format of the list is: {COBOL 68 construct} =>

{what to do with it}

o   SELECT... ASSIGN TO <name> => replace the <name> by

the actual title in quotes.

The following list shows the filename changes for the IMCSRS programs. The first column shows the internal filename, i.e. the FD name, where "x" means any letter. The second column is the name we used in the ASSIGN TO <name> to match the test data filenames. And the third column shows what the name should be in order to use the RMplusDB product. (The problem is that relational databases, like Oracle, do not allow table names with embedded periods.) It is possible to use environment variables to map the "New External Filenames" to the "External for RMplusDB" names so that you do not have to modify the source, but this seems like it could be very clumsy.

Internal FILENAME	New External FILENAME	External for RMPLUSDB
xB01AGU IC01AGU 0C01AGU	"MB01AGU.DAT" "MC01AGU.DAT" "0C01AGU.DAT" (rename to MC01AGU.DAT after generating)	"ECCLIN" "UIC"
xA03AGU	"DG01AGU.DAT"	"UICTRAN" "ECCLINTRAN"
xG01AGU	"DG02AGU.DAT"	"CONTROL"
IK03AGU	"DK03AGU.DAT"	

o   RECORDING MODE IS F => delete T

This clause is not supported on RM/COBOL85 and isn't needed.

o   EJECT => delete

This causes the program listing to go to top-of-form on the mainframe. This clause is not supported on RM/COBOL85 and isn't needed.

o   EXAMINE => INSPECT

This is a case where COBOL 68 uses a different keyword from COBOL 74 or 85.

APPENIDX B COBOL TO SQL REPLACEMENT EXPERIENCE

BRET JOHNSON

This appendix describes a partial reverse engineering of the P13AGU IMCSRS COBOL program. What follows describes my experience in providing the same functionality via SQL.

I chose a COBOL program, P13AGU, that Gary has ported and began trying to convert it to completely to Oracle. This program generates an equipment availability report. To understand this program's function, I studied the technical report[12] and the program's source code. I made the following observations:

o   Studying the technical report helped me much more
    that studying the program source. Part of the
    reason for this is that my knowledge of COBOL is
    very limited. However, with the help of a couple
    of books on COBOL, I was able to understand many
    of the low-level aspects of the program fairly
    well. I seemed, though, to have a lot of trouble
    understanding at a higher level what the program
    was doing.

o   Particularly troubling was the frequent use of
    abbreviations in the program, especially in the FD
    fields. Whenever I learned, from the technical
    report, what an abbreviation stood for, I felt
    that my understanding of the program significantly
    improved. I still, however, do not know what
    several FD field name abbreviations stand for. I
    began accumulating a glossary of abbreviations and
    what they stand for. I think such a glossary
    would be of great help to reverse engineers.

o   Also troubling was the lack of sample data files,
    the lack of sample reports that the program gen-
    erates, and my inability to actually run the pro-
    gram and see what it does. All of these things
    would be of help to reverse engineers.

o   It was clear that the program used two entities --
    date cards and 2406 transactions. However, I
    couldn't quite understand the program well enough

to understand the relationship between these two entities.

With the knowledge I did obtain, I began porting the program to Oracle. I constructed two tables based on the FDs, created some sample data, and used the Oracle loader to load it into one of the tables. I learned, mostly from the technical report, how the equipment report is structured (making some guesses to fill in my missing knowledge), and generated such a report in Oracle.

There appears to be three ways to do reports in Oracle:

o   Just do an SQL query.

o   Use SQL*Plus to format the results of an SQL query

in various ways (i.e., specify groupings, generate

totals for an attribute on each group, label

column headers, and so on).

o   Use SQL*ReportWriter to do fancier reports than

those supported by SQL*Plus. According to the

documentation, the advantages of SQL*Report over

SQL*Plus are that SQL*ReportWriter supports

reports requiring more than one SQL statement and

supports nontabular reports, such as checks.

I used the second option for my report. Specifically, I went through the following steps when implementing some of P13AGU's functionality in Oracle. I created the following two tables in Oracle, corresponding to the two FDs in the program. I created these tables by directly constructing the SQL CREATE command, not by making an ER diagram with STP and having STP construct the SQL CREATE command. Note that in Oracle one can use the GET command to load a file containing an SQL command into the SQL buffer and the RUN command to execute the command in the SQL buffer.

69

```
┌─────────────────────────────────────────────────────────────┐
│                        DATE TABLE                            │
├─────────────────────────────────────────────────────────────┤
│                                                              │
│  create table date_card        (army_area char(1),          │
│                                 cutoff_day char(2),          │
│                                 cutoff_mon char(3),          │
│                                 cutoff_yr number(2),         │
│                                 cutoff_jl day number (3),    │
│                                 rpt_beg_day char(2),         │
│                                 rpt_beg_mon char(3),         │
│                                 rpt_beg_yr char(2),          │
│                                 rpt_beg_jl_day char(3),      │
│                                 station_name char(34),       │
│                                 ctl_edit char(1),            │
│                                 card_code char(1))           │
│                                                              │
└─────────────────────────────────────────────────────────────┘
```

```
┌─────────────────────────────────────────────────────────────┐
│                        2406 TABLE                            │
├─────────────────────────────────────────────────────────────┤
│  create table t2406                                          │
│                                (uic char(6),                 │
│                                 seq char(3),                 │
│                                 nomen char(8),               │
│                                 model char(10),              │
│                                 ecc_lin char(8),             │
│                                 auth char(3),                │
│                                 oh char(3),                  │
│                                 poss number(5),              │
│                                 aval number(5),              │
│                                 o_sup char(5),               │
│                                 o_maint char(5),             │
│                                 s_sup char(5),               │
│                                 s_maint char(5),             │
│                                 util char(1)                 │
│                                 card_code char(1),           │
│                                 nomen_p char(10),            │
│                                 or_p char(2),                │
│                                 org_name_p char(20),         │
│                                 alo_p char(1),               │
│                                 station_p char(5),           │
│                                 sort_key char(21))           │
│                                                              │
└─────────────────────────────────────────────────────────────┘
```

Note that in the COBOL code, the date FD has its fields defined explicitly as part of the FD in the file section of the program. The 2406 FD, on the other hand, is defined explicitly in the file section as just having a single field, 151 characters long. It is the working storage section, however, where "WS-D-REC" is defined, that indicates that this single 151 character field can be thought of as being broken down into 25 smaller fields. The fact that the 2406 FD and WS-D-REC working storage are related is not made explicit by COBOL syntax anywhere in the file section or working storage section. The following syntactic construct in the procedure division does, however, give evidence that WS-D-REC working storage and 2406 FD (which is officially called the IG09AGU FD) are related:

0427   READ IG09AGU INTO WS-D-REC

0428   AT END

0429   GO TO 0290-FINAL-PROCESSING.

Therefore, an automated tool which constructs ER diagrams from COBOL source might very well need to look in the procedure division to know what attributes to give to an entity. Another hint that the WS-D-REC working storage and 2406 FD are related is that the total of the lengths of the 25 components of WS-D-REC is 151 characters, exactly the same as the length of the single field of the 2406 FD.

I do not know why the original coders did not make the 25 logical fields of the 2406 FD explicit in the definition of the FD. In theory, doing things the way they did would allow them to treat the 151 characters in a 2406 FD record as being broken down in different ways into logical fields. However, it doesn't appear that the P13AGU code ever treats the records in a different way than containing these 25 logical fields.

I then constructed some sample data that could be used to generate a report. Here is the sample 2406 data:

        00000D 12345 20 15 234

        00000A 23105 36 23 456

        00000E 40395 52 29 234

        00000D 10323 34 24 234

        00000C 20333 20 19 456

        00000B 44444 25 17 234

I loaded this table into the t2406 table with the following command:

o load data infile t2406 replace into table t2406 fields

        terminated by whitespace (uic, model, poss, aval, station_p)

I then executed a few commands to format the query results. These commands included TTITLE and BTITLE, to place titles at the top and bottom of each page; COMPUTE, to tell Oracle to compute the average; BREAK, to group records with the same value of STATION_P together; and perhaps other formatting commands. I then executed an SQL SELECT command to list some fields from all of the records in the t2406 table.

The report generated by the above process is the following. Note that this report does not contain any data from the data_card table, just the t2406 table:

Wed Nov 20   page   1

     Inventory Readiness Report

Stati UIC   AVAL   POSS AVAL/POSS

Stati	UIC	AVAL	POSS	AVAL/POSS
234	00000B	17	25	.68
	00000D	15	20	.75
	00000D	24	34	.705882353

```
         00000E  29   52   .557692308              :                      .
*****                          ----------
avg                            .673393665
456   00000A  23   36   .638888889
                   Sample Report
Wed Nov 20                     page   2
      Inventory Readiness Report
  STATI   UIC  AVAL  POSS  AVAL\POSS
  456   00000C  19   20         .95
  *****                          ----------
  avg                            .794444444
                   Sample Report
```

6 rows selected.

Among other information, this report gives the percentage of days that a certain UIC was available. "AVAL" is the number of available days, "POSS" is the number of possible days, and "AVAL/POSS" is thus the percentage of days that a UIC possibly could have been available that it actually was available (at least this was my reasoning). I generated this report based mostly on what I learned from the technical report about what information the Inventory Readiness Report is supposed to contain, using logical guesses to fill in missing pieces. I didn't use my reverse engineering of the P13AGU source code very much to learn about the content and structure of this report, as I couldn't understand it well enough to do so. I believe that if I would have had a sample Inventory Readiness Report in front of me, making my Oracle Inventory Readiness Report would have been much easier and more accurate.

As I studied the technical report and the P13AGU source code, I made up a glossary of what abbreviations stand form. I found this glossary very helpful, since I would often discover what an abbreviation stands for and then forget what it stands for a little later. Here is the glossary:

Glossary	
Term or Acronym	Definition
IMCSRS	Installation Material Condition Status Reporting System
UIC	unit installation code
nomen	nomenclature
model	model/serial number
auth	number authorized
poss	possible days
aval	available days
o_sup	organization supply
o_maint	organization maintenance
s_sup	support supply
s_main	support maintenance
util	utilization code

o EXAMINE... TALLYING => INSPECT... TALLYING TALLY

   FOR and define 77 TALLY PIC 99.

COBOL 68 has a predefined REGISTER called TALLY that the EXAMINE... TALLYING construct updates. In newer COBOLs a program can replace this construct with a program variable named TALLY like any other numeric variable.

o string literals extending past end-of-line => insert

        quote before end-of-line and adjust continuation of

        string on the next line

COBOL 68 assumes that long string literals go all the way to the end of the source line (without a quote mark) and then continue on the next line with a hyphen in column 7 and a starting quote mark for the rest of the literal. COBOL 74 and 85, however, require string literals to have matching quotes around each segment and then automatically concatenates them.

o WRITE... INVALID KEY on sequential file => add

   RELATIVE to SELECT statement

COBOL 85 only allows AT END on a sequential file. To accommodate INVALID KEY clauses, we had to make files RELATIVE, rather than sequential.

o MOVE CURRENT-DATE TO... =>

        01 WS-CURR-DATE.

        03 WS-CURR-YY   PIC 99.

        03 WS-CURR-MM   PIC 99.

        03 WS-CURR-DD   PIC 99.

   and

        MOVE CURRENT-DATE TO...

ACCEPT WS-CURR-DATE FROM DATE.

MOVE WS-CURR-DD TO...-DD.

MOVE WS-CURR-MM TO...-MM.

MOVE WS-CURR-YY TO...-YR.

COBOL 68 has a predefined read-only register called CURRENT-DATE that returns the current date in MMDDYY format. The closest equivalent in COBOL 85 is an ACCEPT... FROM DATE, which returns the current date in YYMMDD format.

o   IF SORT-RETURN... => delete IF statement

This COBOL 68 construct tests for errors on an internal sort. COBOL 85 does not return any sort errors.

APPENIDX C DIFFICULTIES IN USING PCYACC IN A UNIX ENVIROMENT

There were several difficulties that COBOL presented to the use of PCYACC directly on a Unix workstation.

- o First of all, COBOL uses multi-character opera-
  tors, such as ">=". Unfortunately, Unix yacc is
  not capable of dealing with these. There are
  several ways of overcoming this limitation. We
  chose to break the operator in two ('>' '=').*

- o Another difficulty that was simple to overcome was
  a name conflict between several COBOL keywords and
  directives used by NewYacc. We had seen this
  problem previously with other languages and were
  quickly able to make systematic name changes in
  the COBOL grammar.

- o A much more troublesome difficulty arises from the
  number of keywords in the COBOL language. Yacc
  has a statically-allocated table to hold keywords,
  and its size is exceeded by COBOL's requirements.
  We initially overcame this difficulty by making
  use of the bison yacc clone available as part of
  the GNU software distribution. Later, when we
  tried to integrate with NewYacc, we had to abandon
  bison and obtain the source code for the Berkeley
  version of yacc, manually alter the table size,
  and rebuild the tool.

- o One potential difficulty of which we have not yet
  determined the extent is the accuracy of the COBOL
  grammar. Recall that IMCSRS was originally writ-
  ten with an early version of COBOL. PCYACC sup-
  ports a much more recent dialect. Despite our
  efforts to convert the COBOL code, there may still
  be constructs with which PCYACC is not capable of
  dealing.

- o Another difficulty that we did not pursue was the

issue of PARSING CONFLICTS. Yacc is capable of

producing parsers for grammars that are LALR(1).

This is a technical limitation; yacc provides

* This approach slightly violates the language rules by making one token into two. The two tokens can now be separated by whitespace, which would not be legal if the operator were considered as a single token. A better choice is to change the lexical analyzer to detect this situation and return a single lexeme to the parser.

default mechanisms for circumventing language con-

structs that violate the property. There are a

variety of instances of these violations in the

COBOL grammar, and we have not yet checked whether

yacc's default rules mitigate the difficulties

without violating the underlying language syntax.

APPENIDX D AUTOMATIC GENERATION OF ORACLE TABLE CREATION STATEMENTS FROM STPER DIAGRAMS

This Appendix describes an experiment in automatically constructing Oracle table declarations from STP ER diagrams. The ER diagram was manually constructed from a COBOL FD taken from the program P13AGU. The FD is defined in the following statement.

```
0030 FD IA07AGU

0031    RECORD CONTAINS 80 CHARACTERS

0033    BLOCK CONTAINS 0 RECORDS

0034    RECORDING MODE IS F

0035    LABEL RECORDS ARE STANDARD.

0036 01 DATE-CARD-IN.

0037    03 D-ARMY-AREA         PIC X.

0038    03 D-CUTOFF-DAY        PIC X(2).

0039    03 D-CUTOFF-MON        PIC X(3).

0040    03 D-CUTOFF-JL-DATE.

0041      05  D-CUTOFF-YR      PIC 99.

0042      05  D-CUTOFF-JL-DA,  P C 999.

0043    03 D-RPT-BEG-DAY       PiC X(2).

0044    03 D-RPT-BEG-MON       PIC X(3).

0045    03 D-RPT-BEG-YR        PIC X(2).

0046    03 D-RPT-BEG-JL-DAY    PIC X(3).

0047    03 FILLER             PIC X(13).

0048    03 D-STATION-NAME      PIC X(34).

0049    03 FILLER             PIC X(10).

0050    03 D-CTL-EDIT          PIC X.

0051    03 D-CARD-CODE         PIC X.
```

Figure 1: COBOL Code FD IA07AGU

The COBOL code was used to manually construct the ER diagram contained in the following figure.
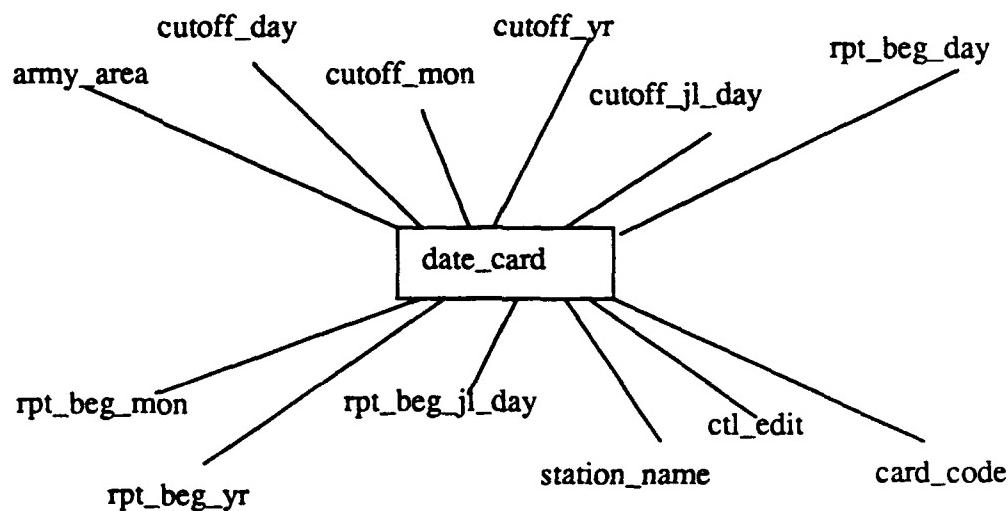

Figure 2: STP Entity Relation Diagram for FD IA07AGU

In addition to the graphical presentation displayed in the ER diagram, STP is capable of retaining annotations describing the properties of the entities being modeled. In our experiment, the annotations were made by hand. It is our expectation, however, that much of this work can be automatically provided by NewYacc and the transformation program described above. The contents of the data dictionary containing the annotations for this diagram is contained in the next figure.

Name: army_area

Defined as Data Element in Entity Relationship

Diagram 'date_card'

Used in 1 Data Structure

date_card

Note AttributeData

Name: army_area

Type: char(1)

Name: card_code

Defined as Data Element in Entity Relationship

Diagram 'date_card'

Used in 1 Data Structure

date_card

Note AttributeData

Name: card_code

Type: char(1)

Name: ctl_edit

Defined as Data Element in Entity Relationship

Diagram 'date_card'

Used in 1 Data Structure

date_card

Note AttributeData

Name: ctl_edit

Type: char(1)

Name: cutoff_day

Defined as Data Element in Entity Relationship

Diagram 'date_card'

Used in 1 Data Structure

date_card

Note AttributeData

Name: cutoff_day

Type: char(2)

Name: cutoff_jl_day

Defined as Data Element in Entity Relationship

Diagram 'date_card'

Used in 1 Data Structure

date_card

Note AttributeData

Name: cutoff_jl_day

Type: number(3)

Name: cutoff_mon

Defined as Data Element in Entity Relationship

Diagram 'date_card'

Used in 1 Data Structure

date_card

　　Note AttributeData

Name: cutoff_mon

Type: char(3)

　　Name: cutoff_yr

Defined as Data Element in Entity Relationship

Diagram 'date_card'

Used in 1 Data Structure

date_card

　　Note AttributeData

Name: cutoff_yr

Type: number(2)

　　Name: date_card

Defined as Entity in Entity Relationship

Diagram 'date_card'

　　　12 Components:

army_area	Data Element
cutoff_day	Data Element
cutoff_mon	Data Element
cutoff_yr	Data Element
cutoff_jl_day	Data Element
rpt_beg_day	Data Element
rpt_beg_mon	Data Element
rpt_beg_yr	Data Element
rpt_beg_jl_day	Data Element
station_name	Data Element
ctl_edit	Data Element
card_code	Data Element

　　Name: rpt_beg_day

Defined as Data Element in Entity Relationship

Diagram 'date_card'

Used in 1 Data Structure   date_card

    Note AttributeData

Name: rpt_beg_day

Type: char(2)

    Name: rpt_beg_jl_day

Defined as Data Element in Entity Relationship

Diagram 'date_card'

Used in 1 Data Structure   date_card

    Note AttributeData

Name: rpt_beg_jl_day

Type: char(3)

    Name: rpt_beg_mon

Defined as Data Element in Entity Relationship

Diagram 'date_card'

Used in 1 Data Structure   date_card

    Note AttributeData

Name: rpt_beg_mon

Type: char(3)

    Name: rpt_beg_yr

Defined as Data Element in Entity Relationship

Diagram 'date_card'

Used in 1 Data Structure   date_card

    Note AttributeData

Name: rpt_beg_yr

Type: char(2)

    Name: station_name

Defined as Data Element in Entity Relationship

Diagram 'date_card'

Used in 1 Data Structure   date_card

    Note AttributeData

Name: station_name

Type: char(34)

    Figure 3: Data Dictionary for Diagram Describing FD IA07AGU

After the diagram is annotated, the STP schema generation capability was used to generate an Ora-

cle SQL command to create a corresponding table. The generated table is given in the following figure.

Date Card	
create table date_card	(army_area char(1), ctl_edit char(1), card_code char(1)) cutoff_day char(2), cutoff_mon char(3), cutoff_yr number(2), cutoff_jl day number (3), rpt_beg_day char(2), rpt_beg_mon char(3), rpt_beg_yr char(2), rpt_beg_jl_day char(3), station_name char(34),

Figure 4: Data Dictionary for Diagram Describing FD IA07AGU

Finally, the command was executed using the Oracle RDBMS and successfully executed.

Some difficulties were encountered in this process:

o   STP allows a variety of types of annotations to be attached to an attribute in a entity-relationship diagram. Besides type information, STP also allows constraints, aliases, key status, author, date/time of generation, and other pieces of information to be attached to an attribute. However, for most of these kinds of annotations, STP does no checking to ensure that what is entered is valid. For example, the user can just as easily enter "ABC#$%" as the type annotation of an attribute (clearly an illegal type in Oracle) as he can enter "NUMBER" (a legal type in Oracle). This

82

lack of checking causes difficulty in that the
reverse engineer does not know until later in the
process, when creating tables, if the type annota-
tions he entered are valid.

o   Another difficulty in the conversion process was
naming rules. Oracle does not allow a "-" (minus
sign) to appear in a name, while "-" appears fre-
quently in COBOL identifiers. This problem was
solved by simply replacing all "-" symbols with
"_" (underscore). Of course, this could be done
automatically by NewYacc.

o   The "date_card" FD contains one group item,
CUTOFF-JL-DATE, made up of two elementary items,
CUTOFF-YR and CUTOFF-JL-DAY. This abstraction is
called aggregation in database circles. CUTOFF-YR
and CUTOFF-JL-DAY are aggregated together to form
a new object type, CUTOFF-JL-DATE. However, this
use of the aggregation abstraction is not sup-
ported in STP's version of the ER model nor in
Oracle. This problem was solved by simply doing
away with the aggregation abstraction.
"cutoff_yr" and "cutoff_jl_day" were made two
attributes of the "date_card" entity in the ER
model and did not represent the fact that these
two attributes can be viewed as a single CUTOFF-
JL-DATE attribute. Of course, this solution to
the problem is not ideal because there is a loss
of information. Note that when attributes are
aggregated together to form an entity, this too is
a form of aggregation abstraction. This form is,
of course, supported by the ER model and Oracle.

The Extended Entity Relationship (EER) model, described in Batini's book [4] supports the aggre-
gation abstraction. Thus in the EER model, CUTOFF-JL-DATE could indeed be represented as the
aggregation of CUTOFF-YR and CUTOFF-JL-DAY. Of course, there is still the problem that Ora-
cle (and most SQL database systems) do not support this abstraction directly.

I find it interesting that many database abstractions that Gary Pardun and I studied in Dr. Navathe's database class, database abstractions that aren't supported by most SQL DBMSs but are supported by research semantic and object-oriented database systems, are also supported by COBOL. An example is the aggregation abstraction described above. Another example is the generalization abstraction that is similar to inheritance in object-oriented systems. An example of the generalization abstraction is when a business has two kinds of employees, salaried and hourly. In an "Employee" table, each of the two kinds of employees could share several common fields but also have several fields that are unique to that kind of employee. This abstraction is supported in COBOL by allowing an FD (such as "Employee") to contain two or more different kinds of records (such as "Salaried" and "Hourly"). I would say that COBOL's support for generalization isn't as elegant as in research databases; the designers of COBOL probably didn't have a clear picture of the generalization abstraction in mind when they designed the language. However, some support is there for it.

The EER model notation also supports generalization. Thus one could reverse engineer COBOL programs to an EER diagram. From the EER diagram, one could go to a modern object-oriented or semantic database system implementation that supports many of the abstractions in the EER model.

However, one could also implement the EER diagram in an SQL database. Batini's book describes how to implement abstractions such as aggregation and generalization, supported by the EER model, in an SQL database system.

APPENIDX E COBOL REPRORTS GENERATED ON THE UNIX WORKSTATION

GARY PARDUN

This appendix discusses the process converting COBOL programs to the workstation environment. In all, the programs p01agu, p02agu, and p03agu were successfully converted and executed. The test data worked nicely after converting to fixed length records, but it consisted of only master file data--no transaction data. So I had to create my own transaction test data. The reports seem reasonable.

I defined and loaded the test data into an Oracle database using SQL*Loader and then did a few simple queries. That worked pretty well too. That exercise convinced me, though, that embedding SQL code in the existing programs would be a difficult and tedious approach. The main problem is that the existing system is so BATCH oriented and SQL is so INTERACTIVE oriented. Much of what the existing programs do is edit-checking batch input data, which would be a very tedious job in SQL. The better, and easier, approach would be to redesign the system based on interactive transactions rather than relying on offline creation of batch disk files. Doing editing in a screen form is much more effective and simple. Any transactions that passed the screen form editing could be written to a transaction table and actually applied to the database later collectively. The RMplusDB product, if it works as we expect, would be a viable alternative to the straight port, but I don't think embedding SQL in Cobol makes much sense.

I also spent some time with RMCo*, the integrated environment, and had some success with it. The main problem is the keyboard interface, which undoubtedly works nicely on a PC, is clumsy on a Unix workstation. It did compile programs and edit them successfully. The debugger seems to work also.

85

References

1. AUTOMATED DATE SYSTEMS MANUAL INSTALLATION MATERIAL CONDITION STATUS REPORTING SYSTEM IMCSRS FUNCTIONAL USERS MANUAL, Commander FORSCOM, AFLG-RO, Ft. McPherson, Georgia, April 1, 1984.

2. "Answer Phase I - Final Report," USAISEC (ASQBG-1-89-027), July 1989.

3. AIRMICS, PERFORMANCE WORK STATEMENT - ISA 97 COMPLIANT ARCHITECTURE TESTBED (ICAT), 1991.

4. Carlo Batini, Stefano Ceri, and Shamkant B. Navathe, CONCEPTUAL DATABASE DESIGN AND ENTITY-REALTIONSHIP APPROACH, Benjamin Cummings, 1992.

5. Peter Chen, "Entity-Relationship Approach to Data Modeling," in SYSTEM AND SOFTWARE REQUIREMENTS ENGINEERING, ed. Richard H. Thayer and Merlin Dorfman, pp. 238-243, IEEE Computer Society Press, 1990. IEEE Software, Volume 1, Number 1, January 1984, pp. 75-88

6. Elliot J. Chikofsky and James H. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy," IEEE SOFTWARE, vol. 7, no. 1, January 1990.

7. Reginald L. Hobbs, Joseph J. Nealon, and Richard Wassmath, "Ada Transition Research Project (Phase I) Final Report," ASQB-GI-91-005, AIRMICS, December 10, 1990.

8. Bret Johnson, "Reverse Engineering with a CASE Tool," SRC-TR-91-07, Software Research Center, College of Computing, Georgia Institute of Technology, December 1991.

9. Kit Kamper and Spencer Rugaber, "A Reverse Engineering Methodology for Data Processing Applications," GIT-SERC-90/02, Software Engineering Research Center, Georgia Institute of Technology, March 1990.

10. Gary Pardun, "A Proposal for Discovering Entity Relationships in COBOL Applications," SERC-91-08, Software Research Center, Georgia Institute of Technology, Fall 1991.

11. James J. Purtilo and John R. Callahan, "Parse Tree Annotations," COMMUNICATIONS OF THE ACM, vol. 32, no. 12, pp. 1467-1477, December 1989.

12. Spencer Rugaber and Kit Kamper, "Design Decision Analysis Research Project," GIT-SERC-90/01, Software Engineering Research Center, Georgia Institute of Technology, January 28, 1990.

13. Elizabeth L. White, John R. Callahan, and James M. Purtilo, THE NEW YACC USER'S MANUAL, Computer Science Department, University of Maryland.

A Comparative Study of DOD and GOSIP Protocols

George N. Rouskas

Georgia Institute of Technology
College of Computing
Atlanta GA 30332-0280

Abstract: We compare existing military protocol standards to their GOSIP counterparts, in terms of functionality and services. We consider network, transport and application layer protocols. We also discuss some of the network and transport layer GOSIP standards that do not have counterparts in the existing military suite of protocols. We address some of the interoperability issues and how they will affect users and the performance of applications.

1.0 Introduction

The Government Open System Interconnection Profile (GOSIP) defines a set of data communication protocols which allows the exchange of information between users of different applications. The procurement of GOSIP compliant communication hardware and software has been mandated by the U.S. Government [GOSIP88, GOSIP89,GOSIP91]. The GOSIP standard is seen as a way to eliminate system incompatibility problems and to offer alternatives to solutions imposed by vendor-specific implementations. As a result, DOD has made a commitment to make a transition from its own standards to international standards. A major motivation has been the decrease in development, procurement and maintenance cost for products based on international standards [GOSIP91, LIN90]. In addition to that, as international standards evolve, they have reached a point of maturity where, not only do they satisfy the DOD requirements, but they also provide greater capability, overcoming limitations of their DOD counterparts [STALL87]. In this report we compare the DOD protocols to their GOSIP counterparts. Table 1 list the set of international and military standards that we will address in this report. This work is motivated by the fact that during the transition period, systems using the old military standards will coexist and must communicate with systems employing the GOSIP standards. Therefore, it is desirable for users to be aware of the different functionality and services offered by the two systems; in addition, this work will provide the basis for evaluating the relative advantages of various solutions for the problem of interconnecting the two types of systems. In addition, we will discuss some of the GOSIP version 2 standards that do not have counterparts in the old military suite of protocols.

	DOD Standards	International Standards
Application Layer	MIL-STD-1780 File Transfer Protocol (FTP) MIL-STD-1781 Simple Mail Transfer Protocol (STMP) MIL-STD-1782 TELNET Protocol	DIS 8571 File Transfer, Access and Management (FTAM) X.400 Message Handling System (MHS) DIS 9041 Virtual Terminal Protocol (VTP)
Transport Layer	MIL-STD-1778 Transmission Control Protocol (TCP)	ISO 8073 Connection Oriented Protocol Class 4 (TP4)
Network Layer	MIL-STD-1777 Internet Protocol (IP)	DIS 8473 Connectionless Mode Network Protocol (CLNP)

Table 1. Military Standard Protocols and Corresponding International Standard Protocols

The paper is organized as follows. Sections 2 and 3 compare the network and transport layer protocols respectively, in terms of functionality and services. In section 4 we discuss the application layer protocols, as defined in GOSIP Version 2, and their corresponding DOD standards. Section 5 describes some additional GOSIP protocols in the network and transport layer, and section 6 addresses some effects that the transition to OSI standards will have on users of current military standards. Finally, section 7 contains some concluding remarks.

2.0 Network Layer Protocols

Both IP and ISO Connectionless Network Protocol (CLNP) provide a connectionless, or datagram service between hosts. In other words, no logical connection between hosts is set up and there is no guarantee that packets will be delivered successfully. In addition, packets that are delivered may be out of sequence. In what follows we examine how the two protocols address the most important issues involved in providing the connectionless service.

2.1 Addressing

Addressing is an important issue for internetwork operation, since individual subnetworks may use different addressing schemes. Internet addresses (used by IP) and network access service point (NSAP) addresses (used by the OSI standard) provide a global addressing scheme by assigning a unique identifier to each host in the internet. This identifier, or global network address, is usually of the form "Net.Host" where "Net" is a subnetwork address and "Host" specifies a host within the

88

subnetwork. Both addressing schemes use a hierarchical arrangement of addressing domains; that is, the set of all addresses, referred to as the global network addressing domain, is divided into network addressing domains in a hierarchical fashion. Every Internet or NSAP address is part of a network addressing domain, which may be further subdivided to subdomains. However, the domains and hierarchical arrangement used by the two schemes differ in a large degree.

As a related issue, the sender station should somehow be able to determine the "Net.Host" identifier for the destination. Both addressing schemes rely on a directory service that provides this unique identifier.

2.2 Routing

Both source and destination addresses appear in the header of both IP and CLNP packets; routing is then generally accomplished by maintaining routing tables in each station. The tables can be static (with or without alternate routes) or dynamic. Dynamic tables handle error and congestion situations with greater flexibility. Other related services available by both standards include source routing and route recording. If a source selects source routing, it provides the path to be taken within the datagram, as a list of gateways to be visited. If route recording is selected, the gateways encountered by a datagram are recorded.

2.3 Datagram Lifetime

The datagram lifetime control function prevents datagrams from looping endlessly through the network. This serves two purposes: to ensure that a datagram does not consume resources indefinitely and to support transport layer requirements. Both protocols assign a "time to live" to each datagram. The time to live is decremented by each gateway the datagram passes through and by the destination host, while datagram fragments are waiting for re-assembly. One difference is in the way the two protocols treat the time to live parameter. In the IP protocol, the lifetime field is set to some multiple of 1 second. Each gateway subtracts 1 second, treating this field as a hop count. However, during re-assembly at the destination host, the lifetime is interpreted as a unit of time. In the CLNP protocol on the other hand, the lifetime field is expressed as a multiple of 500ms. Each gateway decrements this field by 1 for each 500ms of estimated delay. The standard provides some guidance specifying that it is not necessary to subtract a precise measure of the delay, but rather an overestimate of the actual time taken. Therefore the lifetime is treated as a unit of time, which is useful since the transport entity cannot count hops.

2.4 Segmentation and Re-assembly

Different networks may specify different maximum packet sizes and therefore gateways may need to fragment a datagram before forwarding it to the next network. Datagrams are then re-assembled at the destination node, as dictated by both IP and CLNP. Both protocols use essentially the same fragmentation and re-assembly technique, which requires the following fields in the datagram header: the ID uniquely identifies a datagram; the data length gives the length of the data field; the offset is the position of the fragment in the original datagram, and the more flag indicates whether or not this is the last fragment of the datagram.

Since the protocols do not guarantee delivery, they must deal with fragments that do not make it to the destination. The re-assembly process has to be abandoned. To this end, a re-assembly timer is initialized and an algorithm is used to decrement it. When the timer expires before all fragments are received, the datagram is discarded. The algorithms used by the two protocols differ slightly but it is recommended that the lifetime parameter be taken into consideration. In addition, the ISO

89

standard suggests another option, where the re-assembly time is assigned locally at the destination node, independently to the lifetime field.

2.5 Error Control

In a network that does not guarantee delivery, a datagram may be discarded for several reasons, including buffer overflow, lifetime expiration and bit error. When a datagram is discarded by an intermediate station, the station may return an indication of the reason for discard. CLNP mandates the implementation of the error handling function which uses datagrams of a special format. However, the function is available only if the sending network user selects it.

Errors in networks employing IP are reported by sending internet control message protocol (ICMP) messages. ICMP, a user of IP, is a mandatory protocol for providing feedback about problems in the communications environment.

2.6 Flow Control

Gateways and receiving stations experiencing congestion are allowed to limit the rate at which they receive data. Under IP, flow control packets are sent to other gateways and end stations requesting reduced data flow. This is done by using ICMP messages. Under CLNP, the station activates the optional Congestion Notification function informing the network service user (transport entity) of the problem.

2.7 Other Optional Functions

A security option is available in both standards for stations wishing to transmit security information through the network. The security classification system is defined by the user and is not specified in the standards.

A type of service parameter, present in both standards, is used by the network user to request a particular quality of service. The options that may be specified include delivery reliability, delay and throughput requirements. If selected, this parameter is mapped into subnetwork-specific transmission parameters. It is possible that some subnetworks do not support all transmission services; nevertheless they try to match the available services to the desired service quality. Finally, a priority option, a measure of the datagram's relative importance is provided, although there is a different number of priority levels supported by each protocol.

3.0 Transport Layer Protocols

TCP and ISO Transfer Protocol class 4 (TP4) provide a reliable, connection-oriented service; upper layers do not need to be concerned of the details of the communication facility employed. Both protocols provide this service independent of the type and quality of the underlying network facilities. Therefore they must deal with a variety of network characteristics and capabilities. We now discuss in some detail the important aspects of these protocols.

3.1 Addressing

In terms of addressing, a process using TCP is identified by a port number. A socket is formed by concatenating a port number to an internet address and is unique across the internet. TCP provides services by creating a logical connection between a pair of sockets. OSI standards on the other hand, define transport addresses as transport service access points (TSAP); a TSAP identifies a session entity supported by the transport entity. The transport layer maps the TSAP onto a NSAP, which uniquely identifies an end system.

3.2 Multiplexing

In the TCP view, multiplexing refers to the fact that a single TCP entity may provide services to many processes using TCP. This is generally accomplished by associating processes that use TCP services with ports. The TP4 standard on the other hand, supports a broader notion of multiplexing. In addition to allowing multiple users to employ the same transport protocol (distinguished by TSAP), it may also perform upward and/or downward multiplexing.

Upward multiplexing is defined as the multiplexing of multiple transport connections onto the same network connection. This is possible only if the network connection provides sufficient throughput to accommodate all the transport users. On the other hand, if the network connection cannot provide the quality of service required by the transport connection, a single transport connection can be split among multiple network connections. This is referred to as downward multiplexing. Notice however, that upward and downward multiplexing assume that the underlying network service is connection-oriented. They can only be useful when TP4 makes use of, i.e, an X.25 service.

3.3 Connection Management

Connection management generally consists of three phases, namely connection establishment, data transfer and connection termination. We will discuss data transfer separately in the next section; in this section we address the other two phases.

During connection establishment, a logical connection is set up between two transport users. The purpose of the connection is twofold: First it specifies the characteristics to be used for all data transfers on the connection, such as priority, security, throughput and delay requirements. The characteristics may be negotiable and there may be a period of negotiation prior to a successful establishment. The connection may be rejected if some user defined minimum quality of service cannot be guaranteed. Second it enables the transport entity to maintain state information regarding the connection, such as sequence numbers, which is needed for controlling the data transfer.

Connection establishment and termination becomes a problem when packets can be lost or be duplicated. Both protocols use a three-way handshake for establishing and terminating connections, since it is the only scheme that works properly regardless of lost and old packets.

3.4 Data Transfer

Once a connection is established, data is transported in data units that contain sequence numbers. In TCP, every data byte has an (implicit) sequence number (stream oriented operation), while in TP4 data units are numbered sequentially.

Since data units may be damaged or fail to arrive, the need for a re-transmission strategy emerges. Both protocols use a positive acknowledgment (ACK) scheme. The receiver must acknowledge successfully received units; however cumulative acknowledgment is permitted. In TCP piggybacked ACKs are allowed; this is not the case for TP4, which requires the sending of a special packet. A re-transmission timer is associated with each data unit; if it expires before an ACK is received, the unit is re-transmitted. The protocols recommend the use of dynamic timers.

A transport entity that cannot keep up with the flow of data units may regulate the rate at which data arrives by activating a flow control mechanism. This mechanism should be capable to overcome problems arising due to transit delay and lost packets. To this end, both standards use a credit-allocation scheme. In TP4, initial credit is set during connection establishment and additional credit

91

is granted with ACKs. The window of credit is not allowed to be reduced. TCP uses a more general scheme in which the window may be reduced; since packets allocating credit may arrive out of sequence or be lost, the potential exists for the sender and receiver to have a different idea about what the actual value of the credit window is. Finally, we note that duplicate packet detection is done by using the time to live field discussed in the previous section.

3.5 Special Capabilities

Both protocols provide a means for informing the destination transport entity user that significant data are carried by the incoming data unit. This is accomplished using the urgent data signaling mechanism in TCP and the expedited data transfer function in TP4. Urgent or expedited data units are marked appropriately, although they are transferred using the normal network data transfer service. Therefore, these services do not guarantee faster delivery; they only inform the transport user that the arriving data might require a special action.

In addition, TCP supports the data stream push capability. In general, TCP will wait until a sufficient amount of data has been accumulated before it constructs a data unit for transmission. However, the user has the option to set a flag to indicate that all outstanding data be transmitted immediately. On the receiving end, when a data unit marked with this flag arrives, all buffered data up that point are delivered to the user.

3.6 Error Reporting

When the transport entity faces a catastrophic condition from which it cannot recover, it will report to the user indicating that a failure has occurred. TCP provides a special primitive for reporting errors. TP4 on the other hand, issues a disconnect request in which the reason for the failure is included.

4.0 Application Level Protocols

In this section we discuss the application layer protocols, namely FTAM, MHS and VTP, and their DOD counterparts, FTP, SMTP and TELNET. In Figure 1 we compare the DOD and OSI communications architectures. Notice that in the case of the DOD four-layer model, application processes are direct users of the transport entity, while in the OSI model the services available to the application user are enhanced by the presentation and session layers. As we will see, this difference has a major impact on the functions and services supported by the various protocols.

OSI Architecture	DOD Architecture
Application **FTAM, MHS, VTP** **Presentation**	**Process** **FTP, SMTP** **TELNET**
Session **Transport**	**HOST-TO-HOST**
Network	**Internet**
Data Link **Physical**	**Network Access**
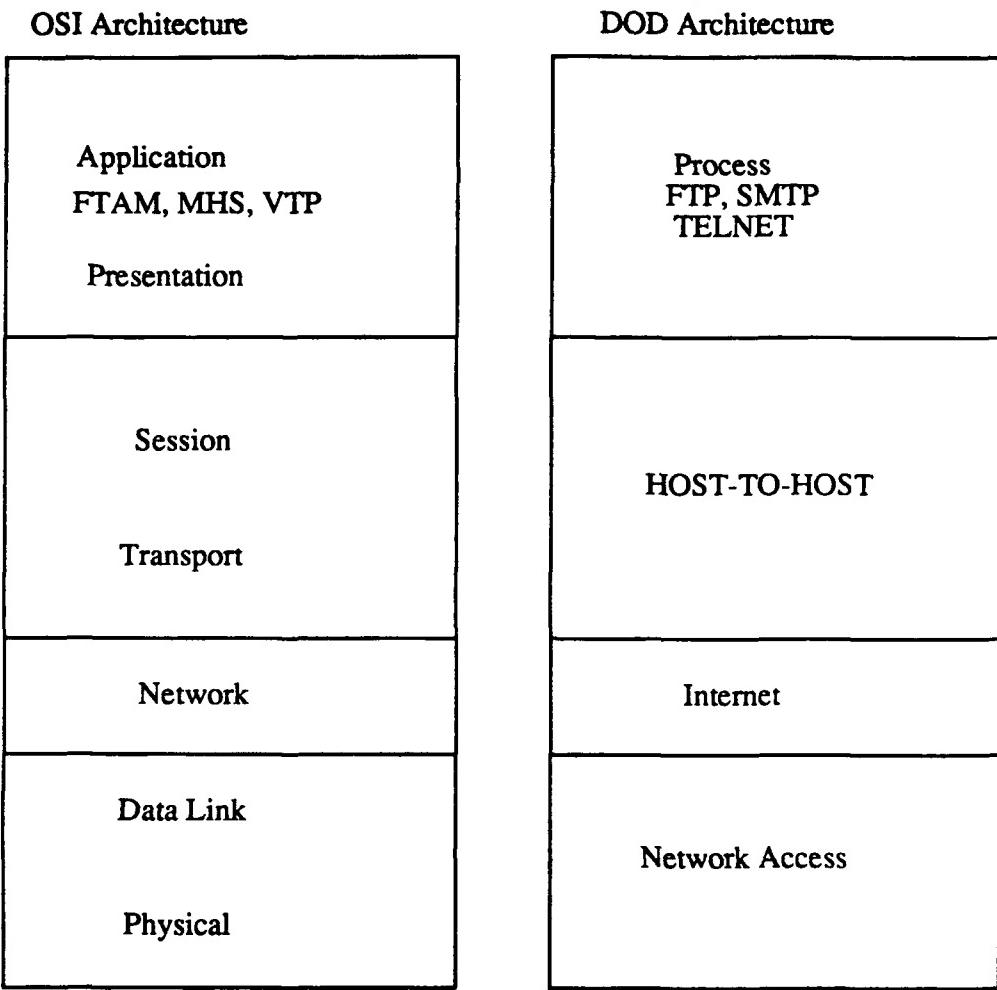
Figure 1. OSI and DOD Communications Architectures

4.1 FTP and FTAM

FTP and FTAM offer a means for exchanging information among computers. Transfer of files is a complicated task since there are almost as many ways to store, access and protect files, as there are computers. How the protocols handle some of the issues involved will be discussed.

FTP provides a way for one computer to send or get a file from another computer. It relies on TCP to provide a reliable data transfer. Two TCP connections, the control and data connections are opened. Control commands and responses are sent over the control connection. This connection manages the data transfer which takes place over the data connection. In terms of functionality, FTP is fairly low-level, since the user process needs to interact with details such as establishment of the data path and the management of the transfer. Access control in FTP is provided during connection setup. It is generally simple and based on login criteria.

FTP does not provide support for every possible data and file type, nor does it define a single virtual file. Instead it provides three dimensions which can be used to provide a common means of infor-

mation exchange between two computers. The first dimension refers to the data type: Text can be represented as either ASCII or EBCDIC characters. All other data are treated as either a bit stream (for example, executable files), or as groups of bits of a certain size. The second dimension refers to the file type: In the simple case, files are assumed to be strings of bytes. However, some applications require that files are organized as a collection of records. In this case, FTP transmits data one record at a time separated by a special marker. Finally, the third dimension, transmission mode, allows the transfer to support transfer checkpoint and restart, as well as data compression.

The OSI file service takes a totally different approach. In order to allow the interconnection of a wide range of systems of different complexity, a common model of files is established, called the virtual filestore. The virtual filestore defines concepts relating the description of files; by agreeing on these concepts, independence from particular file systems is realized. In particular, the virtual filestore defines the following:

The file attributes refer to the properties associated with the file. Attributes are related to the file transfer (filename), the physical storage (creator, date of creation) or for file-related security (access control).

The file structure describes the organization of data in the file. A tree structure is used to represent how units of data within the file are related for access purposes. This structure is capable of modeling most of the existing types of files.

The file operations define the valid operations on files. Operations fall into two categories: operations on the entire file (open, close, read attribute) and operations on the file contents or data units (locate, read, erase).

The file transfer is initiated by the file service initiator and a connection is established with the file service responder on the remote machine. The initiator's authority to access the file is verified during the connection setup, and all control information and data are transferred over the same connection.

4.2 SMTP and MHS

Electronic mail is a facility that allows users at terminals to compose and exchange messages. SMTP and MHS are the protocol standards for transferring mail between hosts.

SMTP is not concerned with the format or content of messages themselves, other than standardizing the character set to ASCII. It assumes that users create messages by using an editor and that with each user is associated a mailbox, a special file where incoming messages are stored. Furthermore, SMTP assumes that messages is placed in an outgoing mail queue.

Each message consists of two parts, the message body and a list of destinations. The queue is serviced by the SMTP sender process which transmits the messages to the proper destination over a TCP connection. The SMTP receiver process on the destination host receives the messages and places them in the appropriate mailboxes, or adds them to the local mail queue if forwarding is required. In addition, it adds timestamp and log information in the start of the message body, indicating the path that the message has taken.

The SMTP sender must deal with a variety of errors, including faulty destination addresses and hosts that are unreachable. In these situations a notification is returned to the sender. In general, SMTP tries to provide reliable operation, by making sure that the destination host has received the message. However, no acknowledgment of a successful delivery is returned and errors are not guar-

anteed to return either. No recovery from hosts that may lose files is provided. SMTP can perform several optimizations. For example, messages that are sent to multiple users on the same host are only transmitted once. In addition, although normally a sender and a recipient are identified by mailboxes, for difficult destinations the destination mailbox can be augmented by a path that specifies how to reach the remote host.

In conclusion, SMTP is similar to, although simpler than, FTP, in that the data exchanged is one of the possible combinations supported by FTP.

The X.400 series of protocols on the other hand, provide a more sophisticated approach for message handling. There is a basic functional distinction between message preparation and receipt, and message transfer. The user prepares and deals with messages under the assistant of functional units called User Agents (UAs). A UA can be primitive, consisting of an editor and a file structure, or can support advanced features such as multimedia editors and text-to-voice conversion. When the message is ready to be sent, the Message Transfer Agents (MTAs) are responsible to deliver it. The originator's UA transfers an envelope containing delivery instructions and the message contents to its MTA. The MTAs provide store-and-forward service between UAs. In addition, three priorities of delivery are supported, and the result can be reported by means of delivery or nondelivery notification, if requested by the sender. Other features include message pick up instead of automatic delivery at the recipient's end and an option for alternative recipient specification.

The standards provide the ability to exchange multimedia messages (including text, facsimile and digitized voice) between all UAs, regardless of their level of sophistication. Therefore, a service to support content conversion is provided; as another option, when content mismatch is identified the mail may not be delivered. Additional capabilities allow a message to be marked as private (in which case an additional password is required) or important, causing a special action to bring it to the user's attention.

4.3 TELNET and VTP

Virtual terminal protocols allow terminals and hosts on a heterogeneous network to communicate, thus providing remote terminal oriented communication between network users. Because of the large number of terminal types that support a variety of characteristics, the general approach is to define a generic representation of terminal characteristics, called a virtual terminal.

The TELNET protocol defines the network virtual terminal (NVT) which is an asynchronous scroll-mode device with unlimited line length and unlimited number of lines. It supports 7-bit ASCII and provides control capabilities associated with common facilities available on TTY-type terminals. TELNET runs on top of TCP and provides two basic services: data transfer and option negotiation. Data is sent as a sequence of bytes. It consists of user data and TELNET commands. Commands are used to provide the NVT control functions. Since some terminals may support capabilities beyond those provided by NVT, TELNET provides option negotiation. If both sides agree on an option, it is put into effect so that they get the best possible service during the connection. Options are not part of the protocol. However, TELNET has its limitations. More specifically, it does not have the flexibility to successfully support new terminal technology; as scroll mode terminals become out-of-date, network users do not enjoy all the capabilities their terminals can offer.

OSI VTP uses a parameterized approach to define the virtual terminal. The services and the virtual terminal characteristics are defined on each connection by a set of parameter values. Users are responsible for negotiating and agreeing upon the set of parameters. In terms of services, the stan-

95

dard supports both synchronous and asynchronous mode of communication. It also provides for delivery control; the user specifies when data is to be delivered. Using this feature, page mode terminals may allow editing before data is delivered to the remote application. The representation and transfer of graphic information is captured by the display object model. A display object is an array of cells of one (one line display), two (scroll mode terminal) or three (page mode terminal) dimensions. One cell contains a single graphic character and has attributes (i.e., color) associated with it. A display pointer points to the current position in the object. Display data is transferred by updating the contents of the display object, changing the attributed of cells, or moving the pointer. In a similar manner, control information is transferred between the users by updating the contents of a control object. The semantics of the control information is not part of the standard since it is specific to terminals. As we can see, VTP offers a flexible mechanism to define virtual terminals and should be able to support new and special purpose terminals.

5.0 Other GOSIP Protocols

5.1 X.25

X.25 is a protocol standard that specifies the interface between an end-host (DTE) and a packet-switched network (DCE). It follows the virtual-circuit approach for managing the transfer and routing of packets; in other words, a logical connection is established before any packets are sent. Two types of virtual circuit are provided: virtual call and permanent virtual call. A virtual call is a virtual circuit which is dynamically established using a call setup and call clearing procedure. A permanent virtual call is a permanent, network assigned virtual circuit. Data transfer occurs as with virtual calls, but no call setup or clearing is required.

One of the most important services provided by X.25 is multiplexing. Up to 4095 simultaneous virtual calls can be supported over a single physical DTE-DCE link. Each packet contains a virtual circuit number that identifies the virtual circuit it belongs to. Other services include flow control using a sliding window protocol, and acknowledgments that have end-to-end (that is, are sent by the receiving DTE), or local (are sent by the DCE or the network) significance. X.25 provides two facilities for recovering from errors. The em reset facility is used to re-initialize a virtual circuit: packet sequence numbers are set to zero and any packet in transit is lost. Recovery is performed by a higher layer. The restart facility is initiated by a serious error condition, and is equivalent to clearing all ongoing virtual calls and resetting all permanent virtual circuits.

5.2 Transport Protocol Class 0

Connection-oriented transport protocol class 0 (TP0) provides the simplest kind of transport connection. It includes only a subset of the services and functions of TP4. More specifically, flow control is not explicitly present in TP0, which relies on network level flow control. Connection release is also based on the release of the network connection. Data units are not numbered, therefore no re-sequencing is possible. Expedited data transfer is not supported and no multiplexing is performed. As we can see, the only services provided are connection establishment and management, data transfer and recovery from protocol errors. TP0 cannot make use of a connectionless network service; a connection-oriented network service must be used instead. In GOSIP version 2, TP0 is used on top of X.25. Both protocols are required for connection to a public messaging system.

5.3 Connectionless Mode Transport Service

ISO has issued a standard for a connectionless transport service (CLTS). This service is not expected to provide ordered delivery of packets, flow control, or error control. Hence, the connec-

tionless transport protocol is minimal; it only provides for data transfer. The transport entity user can specify the desirable quality of service by setting values to parameters such as transit delay, protection, priority and residual error probability. Depending on the value of the last parameter, the transport protocol packet may or may not include a checksum.CLTS is optional in GOSIP version 2 since no GOSIP protocols require it. However, several non-GOSIP protocols can use CLTS for efficiently communicating across local area networks.

6.0 Transition Considerations

Systems based on the GOSIP protocols will replace existing systems based on the old military protocols gradually. During the transition period old and new systems should somehow communicate with each other. Obviously, a host running TCP/IP will not be able to directly "talk" to a host running OSI protocols. An intermediate system that will act as an interpreter is needed; available commercial products that provide this function, and the trade-offs involved, will be discussed in a future report. In what follows, we concentrate on the effects that the transition will have on users and system programmers.

We first notice that, in general, users are unaware of the underlying communication protocols. Their view of the system is in terms of the services provided by the application layer processes. A transition to OSI-related applications will involve more than a transition from one user interface to another. Users should become familiar with the enhanced services offered by the new protocols, overcoming limitations imposed by the old ones. As an example, FTAM allows hosts to exchange files independent of the underlying file structure, storage system or valid file operations, while FTP imposes restrictions to the types of files that can be exchanged. However, the additional functionality will be available only for hosts supporting the OSI protocols. When hosts using different protocols communicate (through some intermediate system), the overall service will be defined by the application protocols that provide the lesser functionality, namely the current military protocols. Observe that, depending on the type of intermediate system, users may be required to know what protocols are supported by the host they need to communicate with.

System programmers on the other hand, usually rely on the transport protocol functions and mechanisms for establishing and managing connections. Knowledge of the network and transport layer services is then necessary, since these services define the network environment. The transition to the OSI model will have a major impact on system programmers. Although TCP/IP and TP4/CLNP offer equivalent functionality, the mechanisms employed are different. As an example, in TCP data transfer is stream oriented while in TP4 it is not (see section 3.4). In addition, protocols with totally different functionality, i.e., OSI transport class 0, connectionless transport service and connection oriented network service, included in, deviate from the conventional TCP/IP model of, and assumptions about the network environment. Therefore, a new approach is required for establishing and maintaining communication.

As another observation, the performance of applications will greatly depend on how hosts supporting different protocols are interconnected. In general, there are two approaches to providing interconnection. The first requires that a gateway be employed (in the network, transport or application layer), that will translate packets from one protocol to another. In the second, each new host added to the network will have to support both protocol stacks (dual stack approach). In a later report currently available commercial products will be discussed in detail. We emphasize however, that performance issues are extremely complicated in nature.

To see this, consider the gateway approach. The lower the layer the packet conversion takes place,

97

the better in terms of delay (computation time at the gateway), since the packet does not have to be processed by many layers of the protocol stacks. However, if the conversion is made at a higher layer, the gateway has more information about the packet and the connection it belongs to, than if the conversion is at a lower layer, and thus it can perform optimizations that may result in a lower overall delay and/or better service.

Another issue related to the performance of applications is that there may exist points in the network that may become a bottleneck. For example, in the dual stack approach, a node employing both protocol stacks may become overloaded since all the communication between GOSIP and non-GOSIP systems has to go through this node. Also, the fact that multiple types of packets will be transmitted on the network, for example IP and CLNP packets, has to be taken into consideration. Routers have to be able to distinguish between the different types of packets and handle each one accordingly. They should also have some additional information, such as which routers can accept what types of packets, for forwarding purposes.

We have mentioned some problems that will arise during the transition period. How much the performance of applications will be affected is difficult to predict. It becomes even more difficult

if there is no actual network that can be studied. Building such a network will be very helpful because it may reveal additional problems that we have not taken into consideration.

7.0 Conclusions

It is generally recognized that IP and TCP provide functionality equivalent to that of their GOSIP counterparts. On the other hand, the functions supported by the DOD application layer standards are also supported by the corresponding GOSIP standards. However, the latter offer enhanced services and functions that are beyond the capabilities of the former. The transition to the OSI architecture and the international standards will overcome the limitations of the military standards and will simplify integration of products and expansion to areas such as document architecture and transaction processing that are not addressed or covered by these standards.

References

Cunningham, I., and Kerr, I. "New Electronic Mail Standards. Telecommunications, July 1985.
U.S. Department of Commerce. "Government Open Systems Interconnection Profile (GOSIP) Version 1". FIPS PUB 146, August 1988.

U.S. Department of Commerce. "Government Open Systems Interconnection Profile Users' Guide Version 1". PB90-111212, August 1989.

U.S. Department of Commerce. "Government Open Systems Interconnection Profile (GOSIP) Version 2". FIPS PUB 146-1, April 1991.

Lewan, D, and Long, G. H. "The OSI File Service". Proceedings of the IEEE, Vol. 1, No. 12, December 1983.

Lini, K. F. "U.S. GOSIP: A Tutorial". Open Systems Data Transfer, August 1990.

McLeod-Reisig, S. E., and Huber, K. "ISO Virtual Terminal Protocol and Its Relationship to MIL-STD TELNET. Proceedings of the Computer Networking Symposium, 1986. National Research Council. "Transport Protocols for Department of Defense Data Networks". February 1985.

Stallings, W. "Handbook of Computer-Communications Standards", Volume 1: The Open Systems Interconnection (OSI) Model and OSI-Related Standards. Macmillan, N. York, 1987.

Stallings, W. "Handbook of Computer-Communications Standards", Volume 3: Department of Defense (DOD) Protocol Standards. Macmillan, N. York, 1988.

THIS PAGE WAS INTENTIONALY LEFT BLANK

Transitioning from TCP/IP to GOSIP Protocols: A Product Overview

Russell J. Clark
Mostafa H. Ammar

Software Research Center
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

1.0 Introduction

We are conducting a market survey of currently available OSI/GOSIP products that would be useful in a transitioning network environment. For this particular phase of the project we have limited our focus to products available for Sun, MS-DOS, and MacIntosh systems. This was. considered to be an appropriate starting point for the AIRMICS testbed network. This document describes the options for transition from current TCP/IP to GOSIP protocols with respect to the state of implementation of GOSIP within the computer industry. This is a draft document based on information gathered to date.

2.0 Transition Options

Several alternatives for transition from TCP/IP to GOSIP protocols have been discussed in literature from the U.S. Government as well as product briefs from communications vendors. These alternatives include ideas such as Application Level Gateways, Dual Protocol Stacks, and Transport Layer Multiplexing. In most cases, a network in transition will include more than one of these alternatives in order to meet the functional requirements of the network users.

2.1 Application Level Gateways

An example of an Application Level Gateway is one that provides connection between TCP/IP SMTP and OSI MHS. This gateway allows users of current SMTP programs, like UNIX Sendmail, to send messages directly to users of a MHS X.400 mail system. Another such gateway product connects TCP/IP FTP and OSI FTAM protocols. This allows transfer of files between TCP/IP and OSI sites. The main design goal of gateways is to provide interconnection with minimal impact to the user. A user of a current TCP/IP system can continue to use SMTP and FTP to communicate with a newly added OSI system. The major disadvantage of this approach is that it limits the function of both systems to the intersection of the two. The FTP user cannot take advantage of the increased functionality provided by FTAM and the FTAM user will not be able to use all the FTAM functionality when accessing a TCP/IP only machine. MHS Gateway products seem to be the most

widely available OSI application. Products are currently available from several different vendors including: Alfalfa Software, OSIWare, Retix, Sun, and Touch Communications. These products are primarily designed to run on a single host within a current TCP/IP network. All mail within that network continues to be delivered over SMTP. Mail destined for X.400 systems outside this network are routed through the gateway software.

An FTAM gateway is not as common as the MHS product. There is one available from The Wollongong Group and we expect that others are available but have not yet identified them. This product also runs on a single system within a TCP/IP network and provides transparent access for FTP users to remote OSI systems as well as access from FTAM users to the TCP/IP systems.

2.2 Dual Protocol Stacks

Dual Protocol Stacks provide complete implementations of the TCP/IP and OSI protocols on a single system. Users of this system use the appropriate stack to connect to the target network. If connecting to an OSI network, the OSI applications are used. A dual stack system can be added to a TCP/IP network as a launching point for all OSI communications initiated from that network. A user on this network would telnet to the dual stack system and then initiate the call to the OSI system. An advantage of this approach is that it allows all users on the local network to take full advantage of the features of the OSI protocols. The drawbacks are that users must be familiar with the interfaces of both protocols and that local resources (e.g. login, disk, memory) must be available on the bridge system for all users.

Several vendors supply software support for Dual Protocol Stacks. We have reviewed information on products from Retix, Sun, and The Wollongong Group. This products are all UNIX based and are designed to run in multiuser configurations. Like the MHS gateways, this product would run on a single system within a current TCP/IP network to provide connection to an OSI network.

2.3 Transport Layer Multiplexing

Transport Layer Multiplexing allows OSI upper layer protocols to operate over either OSI or TCP transport layer protocols. This allows one to use OSI applications on all systems independent of which transport layer software is used. This does not necessarily mean that a user on a TCP/IP system can communicate directly with an OSI system. End systems using this system can only communicate with systems using a similar mixed stack. A standard implementation of this option is outlined in Internet RFC 1006.

Application products in this category are available from Sun and The Wollongong Group. This is also a feature of the ISODE (ISO Development Environment) which is an experimental public domain system.

3.0 Implementation Options

Another important aspect of the TCP/IP to GOSIP transition is the method used to acquire the new technology. One option is for an agency to develop internal products which implement one or more of the various transition options. This would result in a high degree of control over the features in the GOSIP products as well as the schedules for development and deployment. However, this approach would require extensive resources in both time and expertise. This approach also generates a potential for incompatibilities when connecting that agencies GOSIP networks to other GOSIP systems.

Another alternative is to acquire GOSIP products entirely from outside vendors. This allows the

Army to select from multiple available products based on product features. It allows the Army to take advantage of the work already being done on implementation and compatibility testing. A major concern with this approach is that the current availability of certified GOSIP compliant products is limited. For many protocols and computer systems there are either no available products or there is only one. In general, the implementation and certification of GOSIP compliant protocols is well behind the schedule for mandating the installation GOSIP compliant systems.

Fortunately, this technology lag is being addressed in the GOSIP standards. The GOSIP User's Guide describes several alternatives for agencies contending with the lack of available GOSIP compliant products. It recommends that agencies work with vendors who are developing GOSIP products and require in procurement contracts that these vendors provide specific plans for GOSIP certification. When considering protocols for a new installation, OSI protocols should be used whenever they are available even if there are no available GOSIP compatibility tests. This should help provide for increased interoperability of those products even before the standards bodies are able to certify the specific products. The GOSIP mandate also allows agencies to apply for a waiver if they require functionality that is not in an available GOSIP product. We propose that the transition strategy should be largely based on acquiring GOSIP implementations from outside vendors. In addition, it is likely that each agency will require some functionality that is not provided by GOSIP products. This functionality can be developed internally but the agency should also work with the vendors as well as the GOSIP committees to encourage them to include these new features in the GOSIP standards.

4.0 GOSIP Vendors

This section provides information about the specific GOSIP products we are evaluating. It is organized by vendor. This section is based on information and the OSI/GOSIP Software Vendor List we previously distributed. As with that document, this information is preliminary based on information we have been able to gather.

4.1 Alfalfa Software, Inc.

Alfalfa Software markets an end user mail product called Poste Electronic Messaging System. This is a fairly complete mail interface system including message management folders, access control, and multipart messages (including non-textual files). It uses the OSF/Motif user interface and runs primarily on Sun UNIX systems.

This product includes a MHS gateway program that provides an interchange between SMTP and X.400. In this way, Poste users can communicate with MHS users on an OSI network. This product could conceivably be used as a gateway for other SMTP mail users though this functionality still needs to be verified. This product has not been certified as GOSIP compliant but the vendor claims to support GOSIP level 1.

4.2 Novell, Inc.

Novell is a leading supplier of LAN technology for IBM-PC compatible systems. These products have traditionally been based on proprietary protocols. Novell has added modules to their networking products which support standard protocols including TCP/IP and more recently, FTAM and X.25.

These OSI modules, known as Netware FTAM and Netware X.25 provided a limited degree of interoperability between Novell Netware and OSI/GOSIP networks. Specifically, the FTAM product functions as a responder only. This allows OSI systems to access files on the Netware fileserver

103

but does not provide access from Netware client systems to OSI services. Novell indicated plans for an X.400 bridge product in 1992. In general, these products will provide limited interoperability for transitioning networks but will not provide the complete GOSIP support necessary for new network installations. These products are not certified as GOSIP compliant.

4.3 OSIware, Inc.

The main business focus of OSIware appears to be in providing consulting services for agencies developing and installing OSI based networks. They do provide an end-user mail product known as Messenger X.400. This is an extensive mail package including message folders, autoreply, timed delivery, certified mail, aliasing, text and binary attachments. The package runs on Sun UNIX over TCP/IP or SunNet OSI. It runs on IBM-PC compatibles over using Novell Netware or dialup connection.

This product claims to support both X.400 and X.500. It is not clear whether this product could be used as an MHS bridge or whether it only provides end-user MHS support. This product has not been certified as GOSIP compliant.

4.4 Retix

Retix is a major vendor of OSI products and services. They have a wide variety of OSI products and they are very active in the development of OSI standards for protocols which are a part of both current and future GOSIP versions. A large part of the Retix business is in supplying source and object code to OSI developers on a wide variety of systems. We are still gathering information on some end-user products which are available for UNIX and MS-DOS systems. We will describe those products in a future version of this document.

4.5 Sun Microsystems

Sun has developed a fairly complete set of OSI protocols for their family of UNIX workstations and servers. These products include SunNet OSI, SunNet X.25, SunNet FDDI, and SunNet MHS. These products are currently available for Sun 3 and Sparcstation systems. There is some question as to how long they will be available for Sun 3 systems though Sun has made no formal announcement regarding future Sun 3 support.

In addition to providing end-user OSI functionality the SunNet OSI product provides an implementation of the Dual Protocol Stack. This allows a single system to provide access to both TCP/IP and OSI. This system can be used in a current TCP/IP network to provide access to external OSI systems. This product also supports Transport Layer Multiplexing by supporting FTAM across TCP/IP connections.

The MHS product provides an Application Level Gateway between SMTP and X.400 mail networks. None of the Sun products have been certified as GOSIP compliant.

5.0 Recommendations

We propose that the ICAT project proceed in this evaluation by acquiring the products that would support a hands-on testbed for some of the previously mentioned transition options. We recommend the use of the Sun OSI products to provide a Dual Protocol Stack for the test network. We also recommend the evaluation of several of the MHS gateway products to determine which is most effective in a mixed hardware/software environment. The ISO Development Environment should be used to provide both a test of Transport Layer Multiplexing and also to provide hands-on experience with OSI software to the members of the ICAT team.

6.0 Conclusions

While the current selection of GOSIP implementations is limited, there are a number of products which support GOSIP level one and portions of GOSIP level two. Despite the current limited availability, the GOSIP mandate has stimulated a major effort within the industry to develop OSI and GOSIP compliant products. These efforts, in addition to those of the various government agencies, will result in systems that provide broad interconnection between TCP/IP and GOSIP networks. This interconnection is important to allow GOSIP to be introduced with a minimal impact on currently installed systems.

TITLE: OSI/GOSIP Software Vendor List

AUTHOR: Russ Clark

I. TE CREATED: Sept 20, 1991   LAST UPDATE: November 3,1991

The following is a list of vendors we are currently researching for OSI product availability. We will continue to make additions as new information becomes available.

Several of these references were originally found in the document:

"A Catalog of Available X.500 Implementations", Directory Information Services Infrastructure Working Group, INTERNET-DRAFT, July 1991

COMPANY NAME

Alfalfa Software, Inc.   Tel: 617-497-2922

185 Alewife Brook Parkway   Fax: 617-876-2523

Suite 4200

Cambridge, MA 02138

PRODUCTS

Poste Electronic Messaging System - Mail user interface for Sun UNIX systems. Provides message management folders, access control, multipart messages (including non-textual files), and OSF/ Motif user interface.

CONFORMANCE STATUS

Not complete.

Supports X.400 mail interchange format.

COMMENTS

This is product is a client/server system. The X.400 component consists of a gateway demon that converts UNIX style mail messages to X.400. This allows Poste users to communicate using UNIX mail locally and use X.400 to connect to remote OSI systems.

COMPANY NAME

Novell, Inc.   Tel: 404-698-8350

1000 Abernathy Road   Fax: 404-698-9285

Suite 190

Atlanta, GA 30328

Michael Powell - Sales   mpowell@novell

PRODUCTS

NetWare FTAM - Allows access of the NetWare filesystem as a virtual filestore by any system which supports FTAM. This product functions as an FTAM responder only.

NetWare X.25 - Performs packet assembly and disassembly functions.

CONFORMANCE STATUS

Not Complete.

NetWare FTAM - Claim conformance to limited FTAM specification.

Testing not complete, being tested by NIST.

COMMENTS

This appears to be a reasonable transition product for current NetWare installations. I don't see this being a GOSIP solution for "new" networks. They indicated that a X.400 product would be available during Summer'92.

COMPANY NAME

OSIware Inc.   Tel:+1-604-436-2922

4370Dominion Street, Suite 200   Fax:+1-604-436-3192

Burnaby, B, Canada V5G 4L7

PRODUCTS

Messenger X.400 Mail Products - Includes message folders, autoreply, timed delivery, certified mail, aliasing, text and binary attachments. Sun OS product supports TCP/IP and SunLink X.25. Includes graphical user interface. PC LAN version supports Novell Netware or dialup connection.

CONFORMANCE STATUS

Not complete.

Supports X.400 (1984 and 1988) and X.500

COMMENTS

The strongest aspect of this product is wide availability on platforms including Sun OS and IBM-PC. This seems to be a complete X.400 product.

OSIWARE also provides a consulting/integrating service called EDI which works with users and developers working with OSI networks.

COMPANY NAME

Retix

264430th St.

Santa Monica, CA 90405-3009

Sales and information: (213)399-2200

FAX:(213) 458-2685

PRODUCTS

Retix OSI Networking Products

Retix apparently does not have end user products. They only   sell products in source code form. These products form the basis for several of the other end user products available from various hardware and software vendors.

COMPANY NAME

Sun Microsystems

2500 Garcia Avenue

Mountain View, CA 94043

Sales and Information: (800)USA-4SUN

FAX# - (508)671-0661

Aaron Masciotra - Sales Rep

PRODUCTS

SunNet OSI - Includes both client and server FTAM services, OSI routing capabilities, and ISODE development environment. Includes Dual-Stack for parallel OSI and TCP/IP support.

SunNet X.25 - Provides connection to public X.25 networks.

SunNet FDDI - Provides support for 100Mbit/S Fiber Optic Token Ring.

SunNet MHS - Provides access to Sun mailtool and other UNIX mail systems. Supports X.400 standards. Supports submission and delivery of ASCII text files.

CONFORMANCE STATUS

Not complete.

SunNet OSI - Based on FTAM (ISO 8571), ACSE (ISO 8649 & 8650), Presentation (ISO 8822 & 8823), ASN.1 (ISO 8824 & 8825), Session (ISO 8326 & 8327), Transport (ISO 8072 & 8073),Network CLNS (ISO 8473), ES-IS Routing (ISO 9542), CONS (ISO 8878)

SunNet X.25 - Based on ISO 8208

SunNet FDDI - Based on ISO 9314

SunNet MHS - Based on CCITT 1984 recommendations of X.400, X.401, X.409, X.410, X.411, and X.420. Conforms to NIST Implementation Agreements NBSIR 87-3674. Provides address translation as defined in RFC 987 and RFC 1026.

COMMENTS

This is the most complete OSI protocol suite available for the Sun platform. The dual-stack support is important for providing the transition to OSI protocols. Feedback from users indicates good performance in these protocols because of the development work done

to incorporate them into the SunOS kernel. The MHS is a gateway product which provides sendmail to X.400 translation. Product is run on a single host within a local network for connection to external OSI hosts.

COMPANY NAME

Touch CommunicationsInc.

250 E. Hacienda Ave

Campbell, CA95008

Sales and Information: (408)374-2500

FAX:(408) 374-1680

PRODUCTS

Touch Communications provides consulting and porting services for users installing OSI communications.

They have one end user product called Worldtalk 400. This is a mail gateway package which runs on a 386/ix machine and provides interfaces from X.400 to QuickMail, Microsoft Mail, Inbox Plus, cc:Mail, Novell MHS, and SMTP/UUCP.

CONFORMANCE STATUS

Not complete.

COMMENTS

This product uses a unique approach to address the problem of connecting various incompatible mail systems including X.400 systems. While this is a useful transition product it is still unclear whether it could be considered as a GOSIP product.

COMPANY NAME

The Directory Project,

X-Tel Services Ltd,

University Park,

Nottingham, NG7 2RD

Telephone: 602 412648

Fax: 602 790278

E-Mail: x500@xtel.co.uk

PRODUCTS

MDUA(Motif DUA) provides a Motif-compliant X-Windows user interface to the X.50Odirectory.

COMPANY NAME

The Wollongong Group, Inc.

1129San Antonio Road

PaloAlto

CA 94303

Sales and Information: 800/872-8649

 Fax: 415/962-0286

Tom Bromm - Sales

PRODUCTS

WIN/OSI Release 2.1 - 386 Unix only product.

Main features include:

LLS provides TP0,2,4, CLTP, CLNP, CONS, COTP, and ESIS ULS provides FTAM, VT, and TSB (Transport Service Bridge) this is a bridge which provides connection service between TCP/IP and TP4. Supports some third party X.25 hardware for 386 systems.

CONFORMANCE

Not complete.

COMMENTS

This is a limited product. It was recently announced (Summer'91) and difficult to find technical information on these products within the company.

COMPANY NAME

3ComCorporation

5400Bayfront Plaza

Santa Clara,CA 95054

Information:Cyndi Jung

(408) 764-5173

cmj@3Com.COM

PRODUCTS

OSI Network and Directory Services software

Dual Stack OSI and TCP/IP software